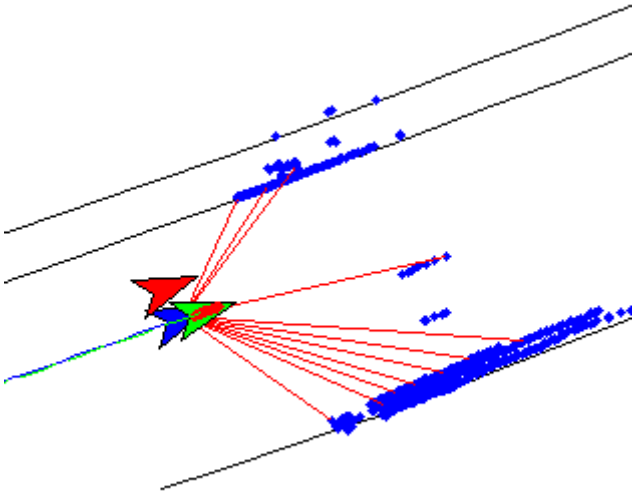

Map Representation and LIDAR-Based Vehicle Localization

Automatic Map Extraction from High Density Point Clouds for LIDAR Localization in Production Vehicles

Master's thesis in the Signal and Systems department, Chalmers

EILIV HÄGG

YINGZHI NING

Department of Signal and System
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis EX024/2016

Map Representation and LIDAR-Based Vehicle Localization

Automatic Map Extraction from High Density Point Clouds for LIDAR Localization in Production Vehicles

EILIV HÄGG
YINGZHI NING

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signal and System
Chalmers University of Technology
Gothenburg, Sweden 2016

Map Representation and LIDAR-Based Vehicle Localization
Automatic Map Extraction from High Density Point Clouds for LIDAR Localization in Production Vehicles
EILIV HÄGG, YINGZHI NING

Supervisor: Robin Lindholm, Volvo Car Corporation
Academic Supervisor: Lennart Svensson, Signal and System department
Examiner: Lennart Svensson, Signal and System department

Master's Thesis EX024/2016
Department of Signal and System
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2016

Map Representation and LIDAR-Based Vehicle Localization
Automatic Map Extraction from High Density Point Clouds for LIDAR Localization in Production Vehicles
Eiliv Hägg
Yingzhi Ning

Department of Signals and Systems
Chalmers University of Technology

Abstract

Within the automotive industry, a lot of resources and effort is put into the development of autonomous vehicles. New functions and driver support systems are added continuously, and in a near future, human interaction for driving will no longer be needed. For this to become reality the cars need to be able to handle all possible scenarios without any accidents or failures. One problem to solve is being able to accurately localize the vehicle in a wide range of conditions. To ensure robust localization, several independent systems that rely on different sensors and maps are needed.

In this thesis, we propose a method for automatic map extraction from a high density point cloud, and a localization algorithm using front-facing LIDAR. The aim of the project is to develop a map representation that contains enough information to allow very accurate localization, with a main focus on lateral positioning, but still being compact in the sense that it requires a low amount of data per unit road. The extracted map together with the localization algorithm, which was based on a particle filter using a likelihood field, were evaluated by running localization simulations using recorded data. The results show accurate estimates of the lateral position with sub decimetre precision, and heading angle with no more than one degree error, but less accurate longitudinal positioning. It was found that the localization algorithm also proved itself robust to surrounding traffic by producing accurate position estimates under short periods of occlusion of the LIDAR.

Keywords: Map Representation, LIDAR Localization, Particle Filter, Likelihood Field, High Density Point Cloud.

Acknowledgements

We would like to give our sincerest thanks to our supervisor Robin Lindholm, who's patience, support, and passion inspired us throughout the thesis. We would also like to thank our academic supervisor Lennart Svensson who was standing by to give us support when we needed it.

The colleagues in sensor fusion team in Volvo cars were very helpful and actively participated in our weekly meetings. Many novel ideas and solutions came from the discussions with them, we would like to express our gratitude to them.

Besides we also want to thank everyone in the Master thesis study group in signals and systems department in Chalmers, including but not limited to Erik Henriksson, Viktor Kardell, Christoffer Gillberg and Kristian Larsson. Many interesting ideas been shared during our discussions and meetings.

Thanks also goes to Volvo car corporation, for financing this thesis and giving us the opportunity to work on it. They provided us access to all the necessary data and tools that we needed to complete the thesis, therefore we would like to express our sincere gratitude to them.

Eiliv Hägg, Yingzhi Ning, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1
Introduction

A future introduction of self driving cars could have a great impact on vehicle safety, mobility, economy, and society as a whole. They could avoid traffic collisions caused by human driver errors, reduce traffic congestion, relieve the driver from navigational chores, etc [1]. For this to become reality the cars need to be able to handle all possible scenarios without any accidents or failures. In order to achieve this it is very important to be able to accurately localize the vehicle on the road, with high precision. To ensure robustness of the localization, several independent systems that rely on different sensors and maps are needed.

1.1 Objective

The goal of this project is two-fold. First, it is to create a method to automatically extract a compact map for LIDAR localization, from a high density point cloud. The map must be compact in the sense that it can be easily stored in the vehicle computer as well as transferred over the cloud, enabling easy download of maps as you go. Second, it is to implement an algorithm that can localize the car in the map using a front facing LIDAR. The localization must estimate three states of the car; lateral- and longitudinal position on the road and heading angle of the vehicle. The main focus is on lateral positioning since the error margin is much smaller there than for the longitudinal position. To guarantee that the vehicle will stay in its own lane, not causing accidents by accidentally switching lane or driving off the road, there is about a maximum of 0.5 m margin on each side of the vehicle, whereas the vehicle could be 2 m off longitudinally without inducing any dangers. The aim for the thesis is that it will be useful for Volvo Cars in deciding which map representation to use and which positioning algorithms to develop in the future.

1.2 Scope

This project will investigate available methods and focus on implementing a complete map representation and positioning algorithm. Only the front facing LIDAR, which is designed for commercialization, will be investigated as the sensor used for positioning. The project is not limited to a real time solution.

1.3 Outline

In Chapter 2, an overview of the data and sensors used in the project, is given. There, specifications of the high density point clouds as well as the front facing LIDAR, can be found. Chapter 3 explains important background theory needed in order to understand the developed methods; basic concepts like homogeneous coordinates and different filtering techniques are described. Chapter 4 is divided into three parts; the first goes through the method for automatic map extraction, the second as well as implementation, optimization and choices regarding the localization algorithm. Chapter 5 presents the results of the project, which are then evaluated and discussed together with a project conclusion in Chapter 6. Chapter 7 describes ideas for future development of the map representation and the localization algorithm. Potential areas of improvement are highlighted here.

1.4 Related Work and Contributions

The mapping and localization problem is a widely studied area. Troughout our thesis we have had a few main sources of information to which we would like to give some extra credit. Probabilistic Robotics [2] by Sebastian Thrun, provides the fundamental concepts of Bayesian filtering. It explains some filters such as extend Kalman filter and the particle filter. This book also introduces the algorithms of different sensor models such as beam-based model and likelihood field model. It also suggests some ways to model measurement noise, which is important for developing positioning algorithms. Several lectures from Introduction to Mobile Robotics and lectures from Robot Mapping at University of Freiburg [3] introduce some essential theory such as homogeneous coordinates and sensor modeling. They explain the basic concepts of homogeneous coordinates and show some examples of how to apply this to Radar and LIDAR data. The concepts and some application examples of clustering methods such as single-link clustering and K means clustering are described in [4] and [5]. Besides these, other literature and articles have been studied throughout the whole project, out of which, some are cited in this thesis.

2
Data sets and Sensors

This chapter explains the data and sensors used in the project. The point clouds and drive-logs used for automatic map extraction are explained and specifications of the LIDAR, used for locaization, are given here.

2.1 High Density Point Cloud

The data for making a compact map was a high density point cloud, made by a third party. It consisted of several merged high precision LIDAR scans, resulting in about 3000-4000 points per square metre road, depending on the environment. Each point had position given in wgs84 latitude, longitude and altitude, which was converted into a local coordinate frame Sweref99TM [6] that is based on metric Cartesian coordinates. The points also had information about reflected intensity. To be able to handle the amount of data in the point cloud, it had been divided up in separate files of approximately 200 meters of road in each. An example of a point cloud can be seen in Figure 2.1.

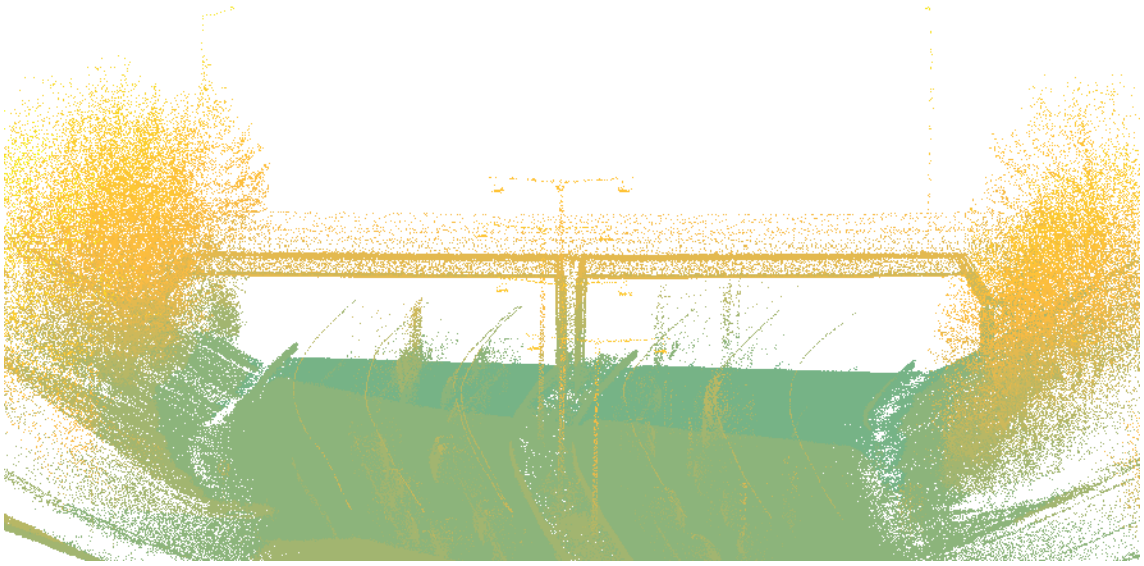

Figure 2.1: An example of the high definition point cloud with artificial color depending on the altitude.

2.2 Barrier Point Clouds

To simplify our work with the map extraction we also had access to data from the high density point cloud where man made barriers such as guardrails etc. had been extracted. The information for each point in the barrier point cloud was the same as in the high density point cloud. Below in Figure 2.2 one can see what parts of the point cloud has been extracted into the barrier point cloud (red).

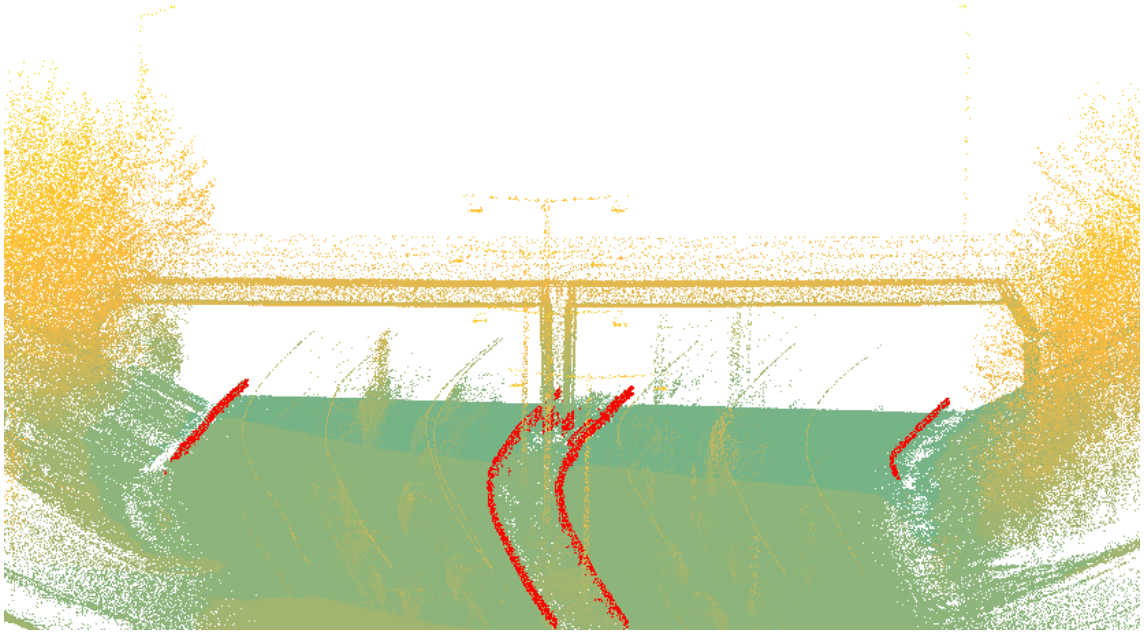

Figure 2.2: The high density point cloud with extracted barriers plotted in red.

2.3 Road Data

Volvo Cars defined an autonomous drive route (AD-route) in Gothenburg, for the Drive Me project, which intends to put 100 self-driving cars on the roads in Gothenburg, Sweden in 2017. In our project, the focus for map extraction has been on a subset of the AD-route which stretches from Gnistängstunneln down to Tynneredsmotet. In Figure 2.3, a map marking the subset, can be seen.

Figure 2.3: A view of the Gothenburg road network, were the used subset of the AD-route is marked in red.

2.4 Drive-Logs

To be able to test and evaluate the barrier map and the localization algorithm, we made use of drive logs from drives along the Gothenburg AD-route. These drive logs contain ego vehicle data such as speed, yaw rate, pitch, roll etc. together with post processed GPS data giving accurate positions for the car with an accuracy of centimetres. This post processed GPS data has been used as ground truth for our localization algorithm.

2.5 Front Facing LIDAR

The front facing LIDAR used in this project, is mounted behind the grill on production cars, facing in the heading direction of the vehicle. The sensor's main purpose is obstacle detection and it forwards information to adaptive cruise control. This is the sensor which has been used for localization in the project. Access was provided to both raw measurements as well as a filtered version of the LIDAR data, as described below.

Sensor specifications:

- infrared laser 905 nm
- horizontal field of view 145°
- horizontal angular resolution 0.25°
- vertical field of view 3.2°
- 4 vertical fields of 0.8° each
- sensor frequency 25 Hz
- distance resolution < 0.1 m
- typical range 150 m

2.5.1 LIDAR Raw Data

The raw data from the sensor consists of a maximum of 5000 readings per scan, where every reading has a given range, horizontal angle and vertical scan layer. With a sensor calibration matrix, the scan data can be converted into Cartesian coordinates in the vehicle ego reference frame. In the LIDAR scans, road side barriers are clearly visible, most of the time. It is also possible to see other vehicles, hillsides, nearby structures etc.

2.5.2 Filtered scan data

In this project we also used a filtered version of the scan data. In the filtered scans, the maximum number of readings have been reduced to 2000 by only keeping the closest reading for two readings in the same angle. The data has also been converted to 2D by recalculating the ranges depending on the vertical angle and thereby discarding the height information. All readings have also been a classified whether it is originating from a static or dynamic object, enabling easy separation of dynamic and static objects. In this version of the scans data, a lot of clutter has been removed and road side barriers as well as on-road vehicles, stand out more clearly.

3
Theory

In this chapter the essential background theory for this thesis is presented. Here, the major concepts will be explained in order for the reader to better understand what has been done in the project.

3.1 Ego Vehicle Reference Frame

The ego vehicle reference frame is a coordinate system with its origin in the centre of the rear wheel axle of the vehicle. This coordinate frame follows along with the car having the positive x-axis pointing in the vehicle driving direction, the positive y-axis pointing to the left and the positive z-axis pointing upwards, see Figure 3.1. This is a useful reference frame for describing objects with respect to the car e.g. sensor data etc. If the position of the car is known objects can be transferred from the ego reference frame to the global frame and vice versa.

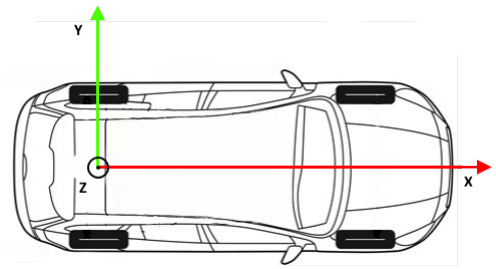

Figure 3.1: The ego vehicle reference frame with origin in the rear wheel axle.

3.2 Homogeneous coordinates

Homogeneous coordinates are a system of coordinates that are commonly used in projective geometry because of how easy they can be transformed between different reference frames and for projections from 3D-space to 2D-space, [7]. These coordinates are also sometimes referred to as projective coordinates. The definition of homogeneous coordinates is that a point $\boldsymbol{x} = \begin{bmatrix} x & y \end{bmatrix}$ is homogeneous if it can be multiplied with any scalar $\lambda \neq 0$ and still describe the same point. Cartesian coordinates $\boldsymbol{x}$ are easily transformed into homogeneous coordinates $\boldsymbol{x}_\mathrm{h}$ by adding an extra dimension that is functioning as a scale factor

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \Longrightarrow \quad \boldsymbol{x}_{\mathrm{h}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{3.1}$$

In this way it can be shown that any homogeneous coordinate that has been multiplied with a scalar still describes the same point in the Euclidean coordinate frame as

$$\boldsymbol{x}_{\mathrm{h}} = \begin{bmatrix} u \\ v \\ \lambda \end{bmatrix} = \begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{3.2}$$

By dividing the homogeneous coordinates with the scale factor and removing the resulting scale factor (which should be 1), one retrieves the corresponding Cartesian coordinates.

By using homogeneous coordinates, affine transformations of points such as rotation and translation can be made by only one matrix multiplication. This is very useful since it can be used to describe the motion of a vehicle or to move points and objects between different reference frames. A transformation matrix $\boldsymbol{M}$ is constructed as

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}, \tag{3.3}$$

where $\boldsymbol{R}$ is a rotation matrix, $\boldsymbol{t}$ is a column vector containing translation information and $\boldsymbol{0}$ is a vector padding the matrix with zero's. In two dimensions the rotational matrix is formed as

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.4}$$

and the translational vector

$$\boldsymbol{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \tag{3.5}$$

where θ is the rotation angle, t_x and t_y are the translation in x and y respectively. A rotation and translation of a point or an object is then be made by

$$\boldsymbol{x}'_{\mathrm{h}} = \boldsymbol{M}\boldsymbol{x}_{\mathrm{h}}. \tag{3.6}$$

The transformation is fully revertible as

$$\boldsymbol{x}_{\mathrm{h}} = \boldsymbol{M}^{-1}\boldsymbol{x}'_{\mathrm{h}} \tag{3.7}$$

and consecutive transformations can done by a matrix product as

$$\boldsymbol{x}' = \boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{x}. \tag{3.8}$$

3.3 Bayesian Filtering

Bayes filter algorithm is a general algorithm for calculating posterior distributions. This algorithm calculates the posterior distribution *bel* from measurement z_t and control data μ_t [8]. Bayes filter is under the assumption that the true states are a Markov model, that is, no variables prior to the state x_t will have influence on the future state, except this dependence is mediated through the state x_t. Bayes filter is also recursive, the posterior distribution $bel(x_t)$ at time t is derived from the distribution $bel(x_{t-1})$ at previous time stamp $t-1$.

Algorithm Bayes Filter $(bel(x_{t-1}, \mu_t, z_t))$
1: do
2: $\bar{bel}(x_t) = \int p(x_t\|\mu_t, z_t) bel(x_{t-1}) dx_{t-1}$
3: $bel(x_t) = \eta p(z_t\|x_t) \bar{bel}(x_t)$
4: return $bel(x_t)$

Table 3.1: Bayes Filter Algorithm

As Table 3.1 indicates a single iteration of the Bayes filter contains a prediction step and an update step. In the second line, the prior distribution $bel(x_{t-1})$ is used to predict the distribution at the current time step. In the third line the measurement z_t is used to update the estimation.

3.3.1 Extend Kalman Filter

The Kalman Filter (KF) is one of the most common Bayes filter implementations due to its low complexity and high accuracy for linear systems [9]. However, KF can only provide high quality results for linear models, while in the real world the state transitions and measurement are usually nonlinear. A widely used approach for nonlinear systems is the Extended Kalman Filter (EKF), which continuously linearizes the nonlinear system and measurement models via Taylor Expansion around the state that is currently estimated, so that the linear KF can be used [10][11].

For each time instance, the posterior distribution is described by the mean μ_t, and the covariance Σ_{t-1}. The EKF algorithm is described in Table 3.2. The first and

Algorithm Extended Kalman Filter $(\mu_{t-1}, \Sigma_{t-1}, \mu_t, z_t)$
1: $\bar{\mu}_t = g(\mu_t, u_{t-1})$
2: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $u_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
5: $\Sigma_t = (i - K_t H_t)\bar{\Sigma}_t$
6: return μ_t, Σ_t

Table 3.2: Extended Kalman Filter Algorithm

second line is the prediction step, where the predicted mean $\bar{\mu}_t$ and covariance $\bar{\sigma}_t$ is

calculated by using the prior distribution μ_{t-1}. Line 3 calculates the Kalman gain K_t at time instance t. This parameter indicates how much the measurement should be incorporated into the new state estimate [12]. Line 4 and 5 shows the update step. It uses the measurement z_t to update the predicted posterior distribution. As one can see in line 4, the mean μ_t is updated in accordance with Kalman gain K_t and the innovation. The innovation indicates the difference between the predicted measurement $h(\bar{\mu}_t)$ and the real measurement z_t. And line 5 shows that the posterior covariance Σ_t is also updated in proportion to the Kalman gain.

However, since EKF only performs well for systems that are almost linear on the time scale of the sampling time, the application for EKF is limited. The first-order linearization could introduce errors in the posterior mean μ_t and covariance Σ_t of the transformed Gaussian random variable.

3.3.2 Particle Filter

The Particle Filter (PF) is a nonparametric implementation of the Bayes filter [13]. The PF estimates the posterior distribution by a finite number of parameters. When using PF, the posterior $bel(x_t)$ is represented by a set of random state samples drawn from the posterior. When illustrating a Gaussian distribution in a parametric form, the probability density function that defines a Gaussian distribution is used. The PF, however, will describe a distribution by a set of particles drawn from this distribution. That is why it is called a particle filter. The PF is nonparametric, which implies that it can be used to describe more kinds of distributions other than the Gaussian distribution as well [13], and it can be used to solve nonlinear problems.

In the PF, particles that are used to represent a posterior distribution can be described as Equation (3.9).

$$\chi_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{3.9}$$

Here M represents the number of particles in the particle set χ_t at time t. Each particle $x_t^{[m]}$ (1<m<M) is a hypothesis of the true state. The goal of the PF is to estimate the posterior distribution $bel(x_t)$ by the set of particle χ_t. The PF algorithm is described in Table 3.3.

Algorithm Particle Filter (χ_{t-1}, μ_t, z_t)
1: $\bar{\chi}_t = \chi_t = \phi$
2: for m = 1 to M do
3: sample $x_t^{[m]}$ $p(x_t \mid \mu_t, x_{t-1}^{[m]})$
4: $\omega_t^{[m]} = p(z_t \mid x_t^{[m]})$
5: $\bar{\chi}_t = \bar{\chi}_t + < x_t^{[m]}, \omega_t^{[m]} >$
6: end for
7: for m =1 to M do
8: draw i with probability $\propto \omega_t^{[i]}$
9: add $x_t^{[i]}$ to χ_t
10: end for
11: return χ_t

Table 3.3: Particle Filter Algorithm

As we can see the PF algorithm constructs the particle set χ_t recursively from the previous time instance particle set χ_{t-1}. Line 3 shows that a state hypothesis x_t at time instance t is generated based on the particle $x_{t-1}^{[m]}$ and the control input μ_t. In line 4, the weight $\omega_t^{[m]}$ of the particle $x_t^{[m]}$ is calculated. It indicates the probability of the measurement z_t given the particle $x_t^{[m]}$. Line 5 indicates the set of particles with their corresponding weights, which represent the posterior distribution $bel(x_t)$.

Line 7 to 10 shows the resampling process, which is an unique character of PF. By resampling, the algorithm replaces M particles from the temporary set $\bar{\chi}_t$ with another set of particles of the same size. The probability the particle $x_t^{[i]}$ gets selected is proportional to the corresponding weight of this particle $\omega_t^{[m]}$. After resampling, particles with higher weight might have been duplicated several times, whereas the particles with the lowest weights might not have been selected at all. The new set of particles will be distributed according the posterior distribution $bel(x_t)$.

3.4 Sensor Models

A sensor model is the environment measurement model of the sensor. It describes how sensor measurements are generated in the real world and how it will be used in the filtering algorithm. The choice of the model depends on the sensor. For the front facing LIDAR it might be described by the beams it sends out and their reflection from the surfaces of the obstacles.

3.4.1 Beam-Based Model

Since a LIDAR measures distance by sending out laser light and counting how long time it takes for the beam to be reflected, the beam-based model is a good fit [14]. The beam-based model is a sensor model for range finders and there are some typical measurement errors of a range measurements that can be incorporated into the

model: small measurement noise, which is usually Gaussian, and random measurement noise[14][3]. Therefore the model $p(z_t|x_t, m)$ should incorporate all these errors.

1. **Local measurement noise**

When the LIDAR measures the range to the nearest obstacle, there might be error within the measurement. This error might be due to the limited resolution of the LIDAR, environment noise on the laser signal, etc. This measurement noise can be modelled as a Gaussian distribution with mean z_t^{k*} and standard deviation σ_{hit}. This noise distribution is denoted as p_{hit}.

Since the LIDAR has limited measured range, the measurement must be smaller than the maximum value z_{max}. The measurement noise can be described by Equation (3.10).

$$p_{hit}(z_t^k|x_t, m) = \begin{cases} \eta\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Here z_t^{k*} is the "true" range of the object measured by z_t^k. $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2)$ is the Gaussian distribution of the noise, which can be calculated as Equation (3.11) shows.

$$\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{(z_t^k - z_t^{k*})^2}{2\sigma_{hit}^2}} \tag{3.11}$$

The normalization factor η can be calculated as Equation (3.12) indicates.

$$\eta = (\int_0^{z_{max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) dz_t^k)^{-1} \tag{3.12}$$

The standard deviation σ_{hit} is tuneable and the optimal value of σ_{hit} could be found by optimization.

2. **Random measurement noise**

There could be random noise in the LIDAR measurement. Random noise can be treated as a uniform distribution across the whole sensor measurement range 0 to z_{max}. The distribution of the random noise is denoted as p_{rand}, and it can be calculated as Equation (3.13) indicates.

$$p_{rand}(z_t^k|x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The local measurement noise and random noise will be added together with its corresponding weights, z_{hit} and z_{rand}, which satisfies $z_{hit} + z_{rand} = 1$. The noise distribution can now be described as Equation (3.14) shows [14].

$$p(z_t^k|x_t, m) = \begin{pmatrix} z_{hit} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{pmatrix} \tag{3.14}$$

The basic assumption for the beam-based model is that the noise in each measurement beam is independent. Therefore the probability can be calculated as the

product of each measurement likelihoods $p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m)$. The beam-based model algorithm is explained in Table 3.4 [14].

Algorithm beam-based model (z_t, x_t, m)
1: $q = 1$
2: for k = 1 to K do
3: compute z_t^{k*} for the measurement z_t^k using ray casting
4: $p = z_{hit} \cdot p_{hit}(z_t^k|x_t, m) + z_{rand} \cdot p_{rand}(z_t^k|x_t, m)$
5: $q = q \cdot p$
6: return q

Table 3.4: Beam-based sensor model Algorithm

This algorithm takes measurement from all beams z_t, vehicle status x_t, and map m as input. As line 5 indicates the probability $p(z_t|x_t, m)$ is the product of all the individual beams likelihood, and this is done by the loop from line 2 to 5. As one can see in line 4, where the individual beam likelihood $p(z_t^k|x_t, m)$ is calculated, both measurement noise and random noise have been incorporated.

3.4.2 Likelihood Field

Likelihood field is another sensor model that might be suitable for LIDAR. The key thought of this model is to project the end points of all the beams of one scan z_t into the same coordinate system as the map. Then find the minimum distance between the end point of the beam and the objects in the map; the smaller the distance the higher probability a beam will get [14]. Like the beam-based model, the algorithm has to consider two types of noises: local measurement noise and random measurement noise.

1. **Local measurement noise**

Similar to the beam-based model, a Gaussian distribution is suitable to describe the local measurement noise. The minimum distance between the end point of the beam $(x_{z_T^k}, y_{z_T^k})^T$ and the objects in the map m is denoted as $dist$. Since ideally the distance is zero, the probability of the beam can be described as a Gaussian distribution with zero mean, as Equation (3.15) indicates.

$$p_{hit}(z_t^k|x_t, m) = \mathcal{N}(dist; 0, \sigma_{hit}^2) \tag{3.15}$$

2. **Random measurement noise**

Just as the beam-based model, a uniform distribution p_{rand} is used to describe random measurement noise.
The probability $p(z_t^k|x_t, m)$ incorporates these two noise distributions, as Equation (3.16) shows, where z_{hit} and z_{rand} are the weights of local and random measure-

ment noise, and $z_{hit} + z_{rand} = 1$. They are tuneable, and the optimized combination depends on the specific situation.

$$p(z_t^k|x_t, m) = z_{hit} \cdot p_{hit} + z_{rand} \cdot p_{rand} \tag{3.16}$$

The algorithm using likelihood field model is illustrated in Table 3.5.

Algorithm likelihood field model $(z_t, x_{z_t^k}, y_{z_t^k}, m)$
1: $q = 1$
2: for all k do
3: if $z_t^k \neq z_{max}$
4: $dist = \min_{x',y'}\{\sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2}| < x', y' >$ occupied in $m\}$
5: $q = q \cdot (z_{hit} \cdot \mathbf{prob}(dist, \sigma_{hit}) + \frac{z_{random}}{z_{max}})$
6: return q

Table 3.5: Likelihood field sensor model Algorithm

As one can see in Table 3.5, the loop is similar to the one in Table 3.4, the final probability is the product of the likelihood of all the end points. The difference is in Line 4 and 5. In line 4, the minimum distance between the end point of a beam $(x_{z_t^k}, y_{z_t^k})$ and the objects in map m is calculated, note that here (x', y') are the points in the map. In line 5, one can see that the final probability is the product of likelihood of all end points. The function $\mathbf{prob}(dist, \sigma_{hit})$ calculates the probability of $dist$ in a Gaussian distribution with zero mean and standard deviation σ_{hit} [14].

3.5 Clustering

There are several different algorithms for clustering data points. In this section two different methods will be described; k-means clustering, which is a very fast algorithm trying to find a given number of clusters, and single linkage clustering which is more computationally heavy but has the advantage that the number of clusters does not have to be known.

3.5.1 K-means clustering

The k-means clustering algorithm is perhaps one the most widely used and studied clustering algorithms [4][15]. Given a set of d-dimensional data points you initialize k cluster centres randomly as guesses of the clusters mean centres, hence the name of the algorithm. The position of these centres are then updated by assigning every point to its nearest centre and then recalculating a new centre from these points. This is then iterated until convergence when no points are assigned to a new centre. The k-means algorithm is very simple to implement and scales linearly with the number of data points. In practice it is a very fast and efficient clustering algorithm, but it has the drawback that it needs to be initiated with the number of clusters k to search for.

Algorithm k-means clustering $(\boldsymbol{x}, k)$
1: initiate k random cluster centres m
2: while not converged
3: assign every data point $\boldsymbol{x}$ to its nearest centre m_j
4: recalculate new centres as $m_j^{\text{new}} = \frac{1}{N_j} \sum_{\boldsymbol{x} \in m_j} \boldsymbol{x}$
5: iterate step 3 and 4
6: return m_j, $\boldsymbol{x} \in m_j$

Table 3.6: K-means algorithm

3.5.2 Single Linkage Clustering

Single linkage clustering, also called nearest neighbour clustering, is a hierarchical clustering method that starts off with every data point being its own cluster [5]. The distances d_{ij} between all the clusters are calculated and the two clusters that are nearest each other are merged into one cluster. This procedure is repeated until all points are in one cluster. The order and distance of merged clusters is usually saved in a dendrogram. By cutting the dendrogram at a certain cutoff distance one gets clusters that that have a minimum distance between each other.

Algorithm single linkage clustering $(\boldsymbol{x}, cutoff)$
1: Calculate all point-to-point distances d_{ij} and store them in an $N \times N$ matrix D
2: Initiate level of clustering $L(0) = 0$ and sequence number $m = 0$
3: Find the two nearest neighbours by $\min(d_{ij})$ in D
4: Merge these two points into a cluster
5: Increment $m = m + 1$ and set $L(m) = d_{ij}$
6: Delete the rows and lines in D corresponding to the two points
7: Calculate new distances between the cluster and all points and insert in D
8: Repeat step 2-7 until only one cluster
9: Trim the clustering tree at desired cutoff distance
10: return clusters

Table 3.7: Single linkage algorithm

The single linkage clustering algorithm has a computational complexity of $\mathcal{O}(n^3)$, so it takes long time for large data sets, but it has the advantage that is does not require a given number of clusters to look for.

3.6 Naive Bayes Classifier

Naive Bayes classifier is a probabilistic classification method in machine learning that is based on Bayes rule [15]. Say you have a data point $\boldsymbol{d}$ that you would like to determine whether it belongs to either class A or class B, where both classes have their own vectors with attributes $\boldsymbol{x}^A$ and $\boldsymbol{x}^B$ defining the two classes by width, height color etc. Then the probability that the point $\boldsymbol{d}$ belongs to, for example, class A can be calculated trough Bayes rule as

$$P(\boldsymbol{d}|\boldsymbol{x}^A) = \frac{P(\boldsymbol{x}^A|\boldsymbol{d})P(\boldsymbol{d})}{P(\boldsymbol{x})}, \tag{3.17}$$

where $P(\boldsymbol{d}|\boldsymbol{x}^A)$ is the probability of the data point given the attribute vector $\boldsymbol{x}^A$, $P(\boldsymbol{x}^A|\boldsymbol{d})$ is the probability of the attributes given the data point, $P(\boldsymbol{d})$ is the prior likelihood of the data point and $P(\boldsymbol{x})$ is a normalizing factor that is the probability of all attribute vectors. With the assumption that all class attributes are independent from each other, the probability of the attribute vector in the numerator of the above equation can be separated as

$$\begin{aligned} P(\boldsymbol{d}|\boldsymbol{x}^A) &= \frac{P(x_1^A|\boldsymbol{d})P(x_2^A|\boldsymbol{d})\dots P(x_n^A|\boldsymbol{d})P(\boldsymbol{d})}{P(\boldsymbol{x})} \\ &= \frac{P(\boldsymbol{d})\prod_{i=1}^{n} P(x_i^A|\boldsymbol{d})}{P(\boldsymbol{x})}, \end{aligned} \tag{3.18}$$

where $P(x_1^A|\boldsymbol{d})$ is the probability of the individual attribute x_1^A given the data point $\boldsymbol{d}$ etc. If one can also assume that the attributes are normal distributed $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ among the data points, the probability product in the nominator can be calculated through

$$\prod_{i=1}^{n} P(x_i^A|\boldsymbol{d}) = \frac{1}{|2\pi\Sigma|} e^{-\frac{1}{2}(\boldsymbol{d}-\boldsymbol{\mu}^A)^T\Sigma^{-1}(\boldsymbol{d}-\boldsymbol{\mu}^A)}, \tag{3.19}$$

where $\boldsymbol{\mu}^A$ is a column vector containing the maximum likelihood estimator mean (MLE) for each attribute and Σ is a diagonal matrix containing the MLE variance for each attribute as

$$\boldsymbol{\mu}^A = \begin{bmatrix} \mu_1^A \\ \mu_2^A \\ \vdots \\ \mu_n^A \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_n^2 \end{bmatrix}. \tag{3.20}$$

If the prior likelihood for all data points are equal, Equation (3.19) will be sufficient to determine which class to assign the point to since the denominator in Equation (3.18), $P(\boldsymbol{x})$, is a constant that is equal for all data points. By calculating the probability of the data point belonging to class A and the probability of belonging to class B, one can classify the point by assigning it to the class with the highest probability. For further explanation of Naive Bayes classifier see [15].

3.7 Forward Selection

Forward selection is a data driven method for selecting relevant feature variables for machine learning classification [16]. The method works in such a way that you start out by evaluating which one of the parameters that performs the best in classifying the data. This parameter is chosen and then you move on to evaluate which one

of the remaining parameters performs the best in conjunction with the previous selected parameter. This is iterated until all parameters have been chose and then you select the set of parameters that performed the best together.

Algorithm forward selection
1: mark all features as available
2: initiate chosen features as 0
3: for (i = 1; i <= number of features; i++)
4: Evaluate classification performance for each available feature together with previously chose
5: Put the feature that performs the best in chosen features and mark it as not available
6: Save classification performance with this feature together with previous features
7: end for
8: select the set of features that together had the overall best classification performance

Table 3.8: Forward selection algorithm

3.8 Cross-Validation

To be able to evaluate a classification algorithm and to avoid overfitting to the data, one usually divides up the available data into a training set and a validation set. The training set is used for training the classifier and the validation set is used to evaluate how well the algorithm performs. In cross-validation the data set is divided into multiple partitions [15], where every part takes turn to function as validation set while the other parts are used for training. The final result will then be the average performance for all validation and training set combinations.

3.9 Twiddle Optimization

The twiddle optimization algorithm is a machine learning algorithm that can be used to tune parameters for optimal performance. It is an algorithm that is based on gradient descent, so it has the disadvantage that it might get stuck in local minima.

Algorithm twiddle optimization

```
1: initiate the parameters with an initial guess
2: initiate a step length dP for each parameter
3: calculate the result of the cost function with the initial parameters, store as lowest cost
4: do while (sum(parameter step length) < threshold and iteration < maxIteration)
5:     for (i = 1; i <= number of parameters; i++)
6:       parameter_i = parameter_i + dP_i
7:       calculate cost through cost function
8:       if cost < lowest cost
9:         lowest cost = cost
10:        step length i = 2·step length i
11:      else
12:        parameter_i = parameter_i − 2 · dP_i
13:        calculate cost through cost function
14:        if cost < lowest cost
15:          lowest cost = cost
16:          step length i = 2·step length i
17:        else
18:          step length i = 0.5·step length i
19:      end if
20:    end if
21:  end for
22:end do while
```

Table 3.9: Twiddle optimization algorithm

4
Method

In this chapter the methods for map extraction and localization are described. The first part goes trough the developed method for automatic map extraction from high density point clouds and the last explains the localization algorithm that uses the resulting map.

4.1 Map Extraction from Barrier Point Cloud

4.1.1 Slicing the Barrier Point Cloud

To be able to handle and process the barrier point cloud in an efficient manner it is divided into smaller pieces. In order to do this, a set of vehicle GPS-poses over the whole demo route is extracted from a drive log-file. The GPS-poses, that are spaced approximately 1 m apart, contains information about x,y,z-position in Sweref99 coordinates as well as pitch, roll and yaw of the vehicle. With this data the whole point cloud can be transferred into the vehicles ego reference frame, see Section 3.1, for one GPS-pose at a time by the transformation described in Section 3.2. For every GPS-pose a "slice", consisting of all the points from one meter behind the pose to one meter ahead in the driving direction, is filtered and saved together with the GPS information of the pose, resulting in many small point clouds of road barriers. See Figure 4.1 for a schematic view.

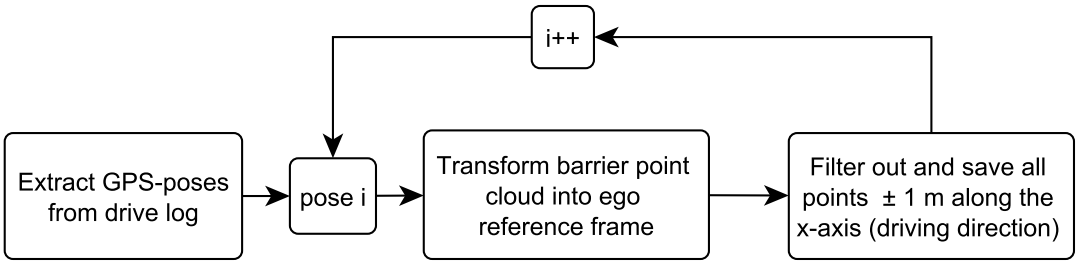

Figure 4.1: Schematic view over barrier point cloud 'slicing'.

4.1.2 Identifying Individual Barrier Segments

To identify individual barrier segments in the resulting point cloud slices from the previous step, each slice is analyzed one at a time by using single linkage clustering,

see Section 3.5.2. Clustering is first run with a rather long cutoff distance of 0.4 m. Resulting clusters that are wider than a threshold of 1 m are clustered again with a shorter cutoff distance of 0.1 m, to see if they are divided up into smaller clusters. These smaller clusters are then considered as individual barriers if they have dissimilar width or height, otherwise they are merged back into one cluster. The reason for this double clustering method is that some barrier segments are very wide and have points too far apart to end up in the same cluster if the cutoff distance is set too low. With the cutoff distance set longer, points that are not in the same barrier segment sometimes get clustered together, however, these are then separated in the second clustering due to having different heights and widths. See Figure 4.2 for a schematic view.

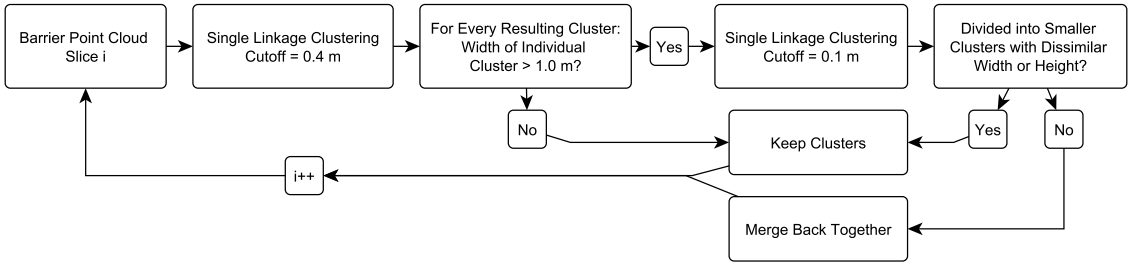

Figure 4.2: Schematic view over barrier point cloud clustering.

4.1.3 Classification and Offset Corrections

In this step the found barrier segments are automatically classified into one of 6 different barrier types, see Appendix A.1, by using a naive Bayes classifier, see Section 3.6 and the section about automatic barrier classification below. To get a robust but easy-to-calculate point describing the position of the each cluster, the mean position of all points in each cluster is used. By calculating the distance and direction from the GPS-pose to each barrier cluster center, one can determine if the barriers are closest to the lanes in the same driving direction or if they are closest to the lanes in the other direction or if the barrier is a center barrier shared by both driving directions. If the distance between two clusters to the left of the GPS-pose is greater than 10 meters this space is classified as the road in the opposite driving direction. If no such distance is found, the road is classified as being behind the barrier furthest to the left. The reason for assigning the barrier clusters to either side is that each driving direction is getting a separate map.

Since the mean centers of the barriers are not necessarily at the distance where they would be detected by the LIDAR sensor, the positions of the barrier segments are plotted together with the logged LIDAR scans to investigate weather any of the barrier types need a lateral offset. By doing measurements of differences between barrier positions and logged LIDAR scans for every barrier type, on several different locations along the road, average lateral offsets for each barrier type can be calculated. These offsets together with the barrier classifications can then be used to place all the barriers center points in the correct position by moving them towards the center of the road for both driving directions. Points in center barriers used by

both driving directions are split into two points, each being moved according to class offset, towards each road center. See Figure 4.3 for a schematic view of classification and offset correction.

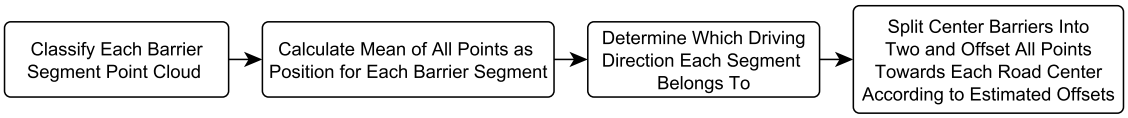

Figure 4.3: Schematic view over classification and offset of barrier points.

4.1.4 Barrier Segment Association

After doing the above steps, one is left with positions of barrier segments that have been classified by type and by witch driving direction they belong to. The remaining problem is now to know witch points to associate with each other as belonging to the same barrier. Since there is information about which point cloud slice the segments originated from, there is no need to search for a best match among all points, but instead one can compare barrier segments from two subsequent slices, since these come in spatial order according to the GPS-poses. At first a search for nearest neighbours between positions of barriers from subsequent slices might seem sufficient, however, at times this results in bridge pillars standing near the barriers sometimes being incorporated in the barrier. To solve this problem the barrier type classification is also included as a parameter. If two points from two subsequent slices are each other's nearest neighbours but have different barrier type classifications, the second nearest point is considered as being in the same barrier if it has the same type classification and is within a distance threshold of 0.5 m. If no such point can be found the two nearest points are associated as belonging to the same barrier despite different type classification, since connected barriers may contain several different type classes. If a mutual nearest neighbour can't be found, the barrier is considered as ended. If a point from the subsequent slice is not associated with any existing barrier it is initiated as the start of a new barrier. This procedure is then iterated over all slices from start to end resulting in sets of positions describing barriers along the road. See Figure 4.4 for a schematic view of the barriers segment association method.

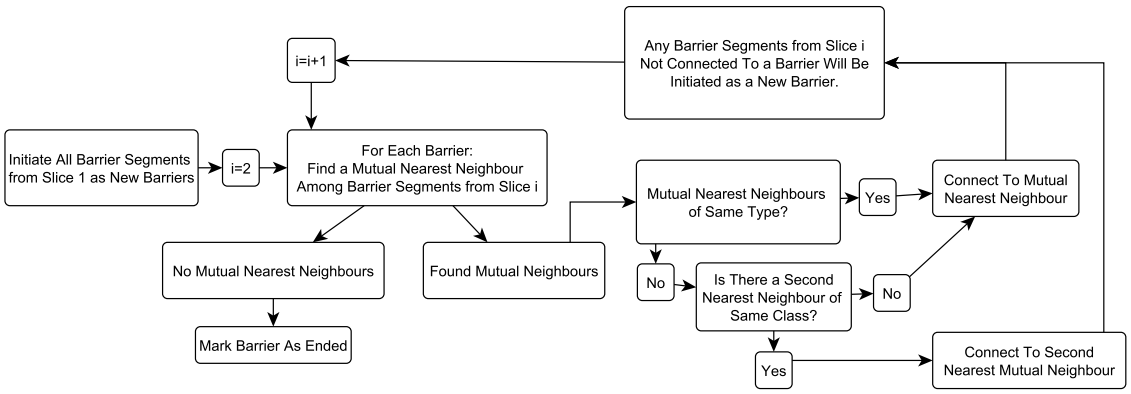

Figure 4.4: Schematic view over the barrier segment association method.

4.1.5 Barrier Corrections

Due to errors in barrier segment classification or errors in clustering, the extracted map has small errors in several places, appearing as meter wide dents in the barriers which are visually conspicuous to the human eye. These errors are corrected on the assumption that the barriers are straight, smooth and have no sudden lateral changes in position. Any point of a barrier that has been classified to another type than the two surrounding neighbours or to a class that is not common among the nearest neighbours, is considered erroneous and its position is recalculated by interpolation of the two nearest neighbours. To check for other errors, all the barriers are searched through by doing a linear regression of four subsequent points and then checking that the fifth point is laying within a threshold of 10 cm from this line. If the next point is laying further away than the threshold, the algorithm searches for a better match along the line, with a maximum search distance of 10 m to the next point. If a point further down the barrier is laying within 5 cm of this line, all the positions of the points in between are interpolated by these two points. If no point is found within the maximum search distance, the barrier is kept as is. In Figure 4.5 you can see a schematic view of the barrier correction method.

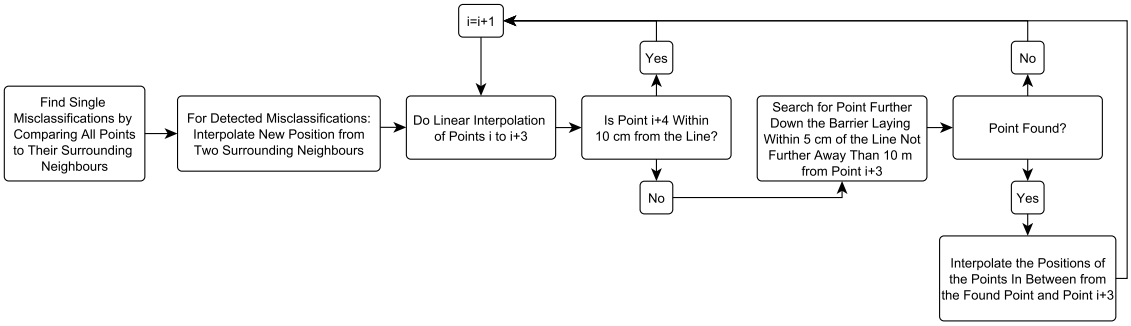

Figure 4.5: Schematic view over the barrier correction method.

4.1.6 Automatic Classification of Barrier Segments

To be able to automatically classify clustered barrier segments to a barrier type, a machine learning algorithm called naive Bayes Classifier is implemented, see Section 3.6 for a description. A training set of barrier segments is made by saving and manually classifying 264 barrier segment clusters from the barrier identification algorithm described in Section 4.1.2. Seven possible feature parameters of the clusters are chosen for evaluation; width, height, y-variance, z-variance, xy-covariance, xz-covariance and yz-covariance. To see which features are best for barrier cluster classification, forward selection is used together with cross-validation. The training set is split up into three parts by randomly assigning barrier clusters to either part, taking one class at a time to be sure to get a few of each class in all three parts. For every evaluation of a new parameter two parts are used for training of the classifier and one part is used for validation. All three parts are rotated to function as validation set and training set, giving a final result of the misclassification rate as the average between the three runs. The forward selection algorithm is iterated until all features have been chosen and the set of parameters used for classification are the set that together gives the lowest misclassification rate.

When training the naive Bayes Classifier for use in the map extraction algorithm the whole training set is used. Since there is little information on how common each barrier type is on the route, the prior likelihood of each type is set equal for all types. The classification is made by assigning the most probable class to every extracted barrier segment according to the probability given by the classifier.

4.1.7 Compact Map Representation

When considering only a meter of barrier at a time, most man made barriers are more ore less straight. Even curved segments are straight enough to be accurately represented by a collection of straight lines. Because of this the choice of map representation is to describe every barrier as a set of connected line segments, all described by a starting point and an end point, both in only two dimensions, x and y. Since the lines are connected, the end point of one line is the starting point of the next, reducing the amount of data even further. Hence, a barrier being described by N lines will only need $N + 1$ points with x and y to describe it.

In the barrier map extraction, positions of the barriers are extracted with one point for every meter. Due to the fact that road side barriers are very straight, a point per meter is in most places redundant, since the result is a set of points all describing more or less the same line. By iterating over every other point in a barrier and forming a line between point k and $k + 2$ and then calculating the distance from point $k + 1$ in the middle to the formed line, point $k + 1$ can be discarded as redundant if it is laying closer to the line than 5 mm. By iterating this procedure over the barrier until no points can be removed, the barrier is now described by fewer points in straight parts reducing the data size of the map significantly. The algorithm for discarding points in this way is described in greater detail below.

Algorithm for detecting and discarding redundant points (*pointsInBarrier*)

```
1: do while nRemovedPoints > 0
2:    nRemovedPoints = 0
3:       for i=1; i<nPointsInBarrier-2 ; i=i+2
4:           form line between point i and i+2
5:           calculate distance from point i+1 to line
6:           if distance < 5 mm
7:              deleteList(end+1) = i+1
8:              nRemovedPoints++
9:       end for
10:      delete all points with indices in deleteList
11: end while
```

Table 4.1: Description of the algorithm that detects and discards redundant points in barrier.

Since the LIDAR often only sees the nearest barrier to the vehicle, barriers that are behind the nearest barrier do not need to be included in the map. This results in a maximum of two barriers along the road in the map limiting the data size even further.

4.2 Localization Algorithm

In order to position the vehicle in the map, it is necessary to choose an appropriate localization algorithm. An easy way to try out different algorithms is to create a simple test environment, with simulated LIDAR measurements and simulated vehicle positions as ground truth. In this way noise free "perfect" data can be achieved. With "perfect" data it easy to determine if a localization algorithm works well since the quality of the data does not effect the result.

4.2.1 Simulation Environment

In order to understand the vehicle localization problem better, a simulation environment was set up in the beginning of the project. The idea was to create simulated data that would not contain any errors or noise. Then implement a particle filter to try to localize the car in a test map. This simulation environment worked as a simple test bench to try out different localization algorithms.

The map, which is largely inspired by an assignment in Chalmers Sensor Fusion course (period 2015/2016, course code SSY320, lead by Lennart Svensson, Signals and systems department), is shown in Figure 4.6.

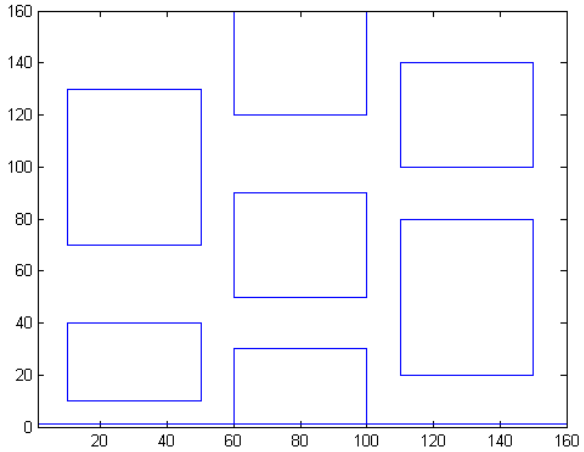

Figure 4.6: Example of test map in simulation environments, this map is largely inspired by an assignment in Chalmers Sensor Fusion course (period 2015/2016, course code SSY320, lead by Lennart Svensson, Signals and systems department)

The route of the car is generated by clicking a path in the map and the resulting path is automatically smoothed. LIDAR measurements for each time stamp is generated according to the position of the car in the route and the map. The sampling time and speed of the car is set constant and the vehicle state x, y, and heading direction θ, at each time instance is calculated accordingly. The LIDAR measurements are generated given the vehicle pose and the angles of the beams. For each time instance and corresponding vehicle pose, the beams of all angles are drawn and the intersection points between beams and the map are determined. The testing environment is illustrated in Figure 4.7, and an enlarged version is shown in Figure 4.8.

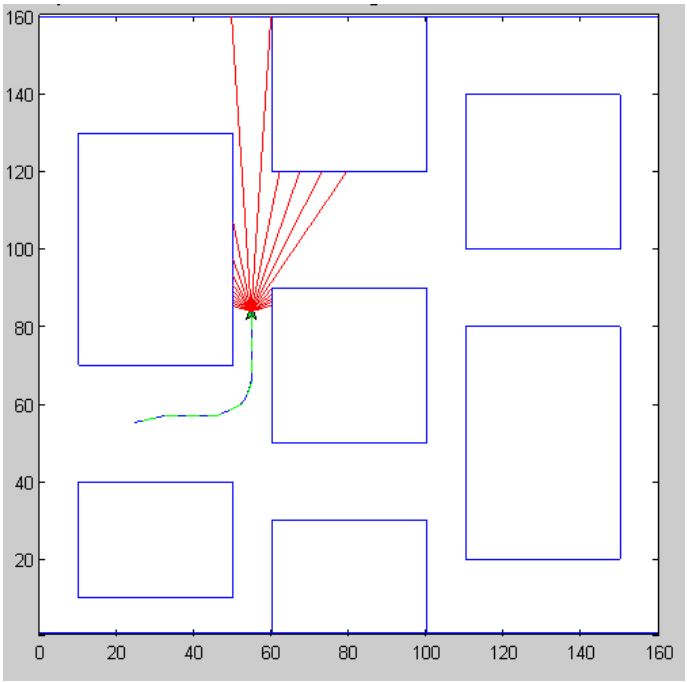

Figure 4.7: Testing environment with route of the car (green and blue dash line) and the LIDAR beams (red lines) shown

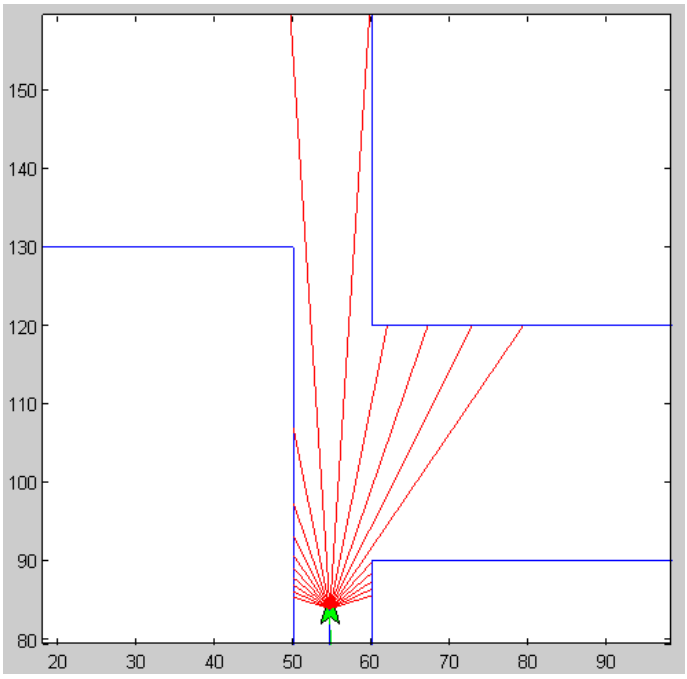

Figure 4.8: Zoom-in bird view of the car (the green object) and LIDAR beams (red lines)

From Figure 4.7 and Figure 4.8 it can be seen that the route of car is indicated by green and blue dashed lines. This route is generated by clicking route nodes in the map. The green object indicates the car, the red lines are the LIDAR beams, and

the length of the beams are the "exact" distance measurements. The LIDAR view is shown in Figure 4.9.

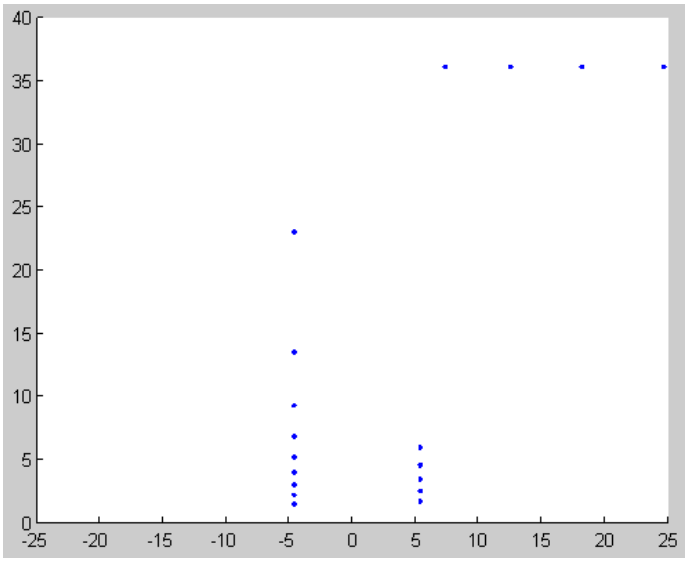

Figure 4.9: LIDAR view when the ego vehicle in (0,0) position and heading upwards

Figure 4.9 shows the artificial LIDAR view. Here there are 20 LIDAR points in total. When the vehicle is standing in the position as the green object in Figure 4.8, Figure 4.9 is what the LIDAR is seeing as the ego vehicle in (0,0) position and heading upwards.

The motion model being used is a simplified description of the car's movement, shown in Equations (4.1) to (4.3).

$$x_{k+1} = x_k + (dT \cdot v_k \cdot cos(\theta_k)) \tag{4.1}$$
$$y_{k+1} = y_k + (dT \cdot v_k \cdot sin(\theta_k)) \tag{4.2}$$
$$\theta_{k+1} = \theta_k + dT \cdot \delta_k \tag{4.3}$$

Here x_k, y_k are the coordinates, θ_k is the heading direction, v_k is the speed and δ_k is the yaw-rate of the car at the time instance k, respectively. The variable dT is the sampling time interval for the particle filter. The assumption is that the car speed and yaw is constant during the sampling period. The speed readings that are used in the motion model are accompanied by Gaussian noise. The standard deviation of this noise is $\sigma_{speed} = r_{speed} \cdot v$, which is proportional to the speed.

The exact yaw rate is calculated as $\delta_k = \frac{\theta_{k+1} - \theta_k}{dT}$, and Gaussian noise is also added to the yaw rate that is used in the motion model. The standard deviation of this Gaussian noise can be expressed as $\sigma_{yaw} = r_{yaw} \cdot \omega + C_{yawNoise}$. As one can see it depends on both the yaw rate and a constant $C_{yawNoise}$, in this way both r_{yaw} and $c_{yawNoise}$ are tuneable.

With the ground truth and measurements set up, a particle filter (PF) is then implemented in order to localize the car in the testing environment. For initial points it is assumed that the initial state is known, therefore all particles are initialized in the same state. The flow of the PF is illustrated in Figure 4.10.

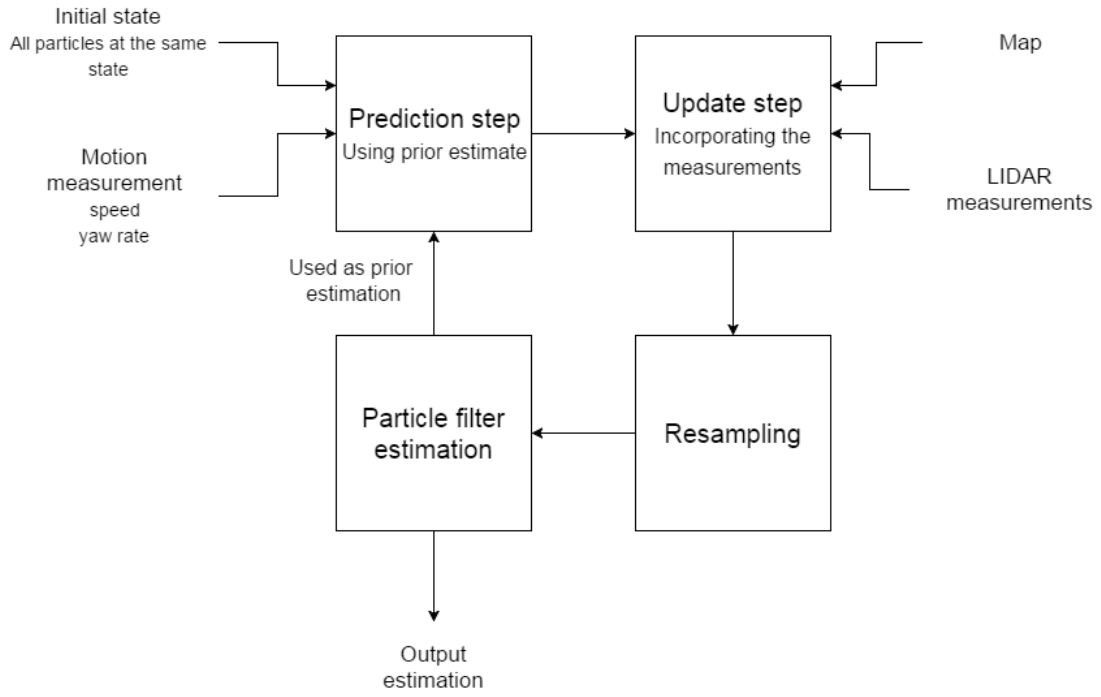

Figure 4.10: Schematic view of the particle filter

With the map, control input (vehicle speed and yaw rate), and LIDAR measurement set up, together with the ground truth, different algorithms can be tested.

4.2.2 Choice of Sensor Model

The front facing LIDAR measures the distance by sending laser light to the object, and reading the reflection from it. With the angle of each beam and its corresponding measured distance known, it is possible to calculate the coordinates of the end point of every beam in the global coordinate frame. In the update step of particle filter, the LIDAR measurements are used to update the weights of the particles, and how to incorporate these measurements depends on the type of sensor model used in the particle filter. Here, two sensor models are investigated: the beam-based model, which compares the end points of every particle state with the LIDAR measurement at the same time instance, and the likelihood field model, which calculates the distance between the end points of particles and the map itself, the smaller the distance the higher the probability. These two models are tested first in the testing environment then in the real world environment with real log data.

For the real LIDAR scan data, there are many scan points per time stamp, and it is too time consuming to take all the scan points into account for update step,

therefore in this thesis only 10 beams are selected as LIDAR measurement for each time stamp. The beams are chosen in such a way that they have a spread over the whole scan. Since a LIDAR scan is performed from left to right it is sufficient to pick every n:th beam to get this spread with $n = m/10$, note that m represents the total number of scan points.

4.2.2.1 Beam-based Model

In the beam-based model, the angle of each beam of the LIDAR is fixed, therefore each particle has its own set of beams given its position and heading angle. For every beam of every particle, the intersection between the beam and the map was calculated. An illustration of the beam-based model in shown in Figure 4.11.

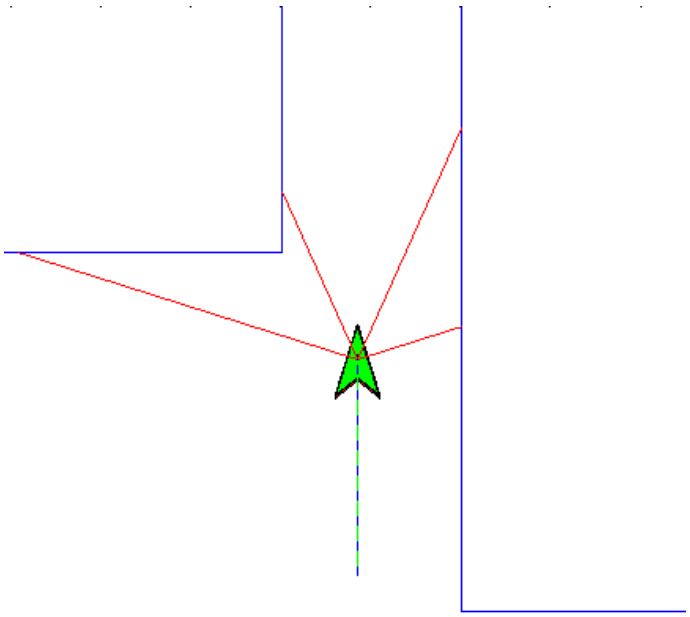

Figure 4.11: Beam-based model illustration

As one can see in Figure 4.11, the blue line indicates the map, the green object represents the car, where the direction it points indicates the heading angle, and the red line indicates the LIDAR beams. Here four LIDAR beams are presented, and their closest intersection with the map are considered as the end points of the beams.

For each particle, every beam has a distance x_t^k according to the position of the particle and hypothetical intersection between map and beam, where $(k = 1, \ldots, K)$ and K is the total number of beams. The measured distances of every beams of one particle is considered as a set $\{x_t^1, x_t^2, ..., x_t^K\}$, and this set of distances is used to compare with the LIDAR measurements, which is also a set of distances $\{z_t^1, z_t^2, ..., z_t^K\}$.

For each beam of a particle, let x_t^k denote the measured distance, and z_t^k denote the LIDAR measurement at the same time instance. The probability of this particle is

calculated according to the algorithm presented in Table 3.4. One can see in line 4 of Table 3.4 that $p = z_{hit} \cdot p_{hit}(z_t^k|x_t, m) + z_{rand} \cdot p_{rand}(z_t^k|x_t, m)$. The parameters z_{hit} and z_{rand} determines the weights of local measurement noise and random noise, and they satisfy $z_{hit} + z_{rand} = 1$. The more weight given to the local measurement noise the more impact the value of standard deviation of the Gaussian distribution will have on the final probability of the particle. On the other hand, the more weight random noise has, the more the final probability will be influenced by the random noise, which is the same for all the beams, that is, the beams will be less distinguishable from each other.

The different combination of z_{hit} and z_{rand} leads to different lateral, longitudinal, and heading angle error of the estimation. For example, with a small standard deviation of the Gaussian distribution of the local measurement noise, the hypothetical beams with a distance that is far away from the true LIDAR measurement is likely to get a very low probability regarding the local measurement noise. In this case if the local measurement noise has a high weight, then more particles are likely to get killed during the resampling step. However, if the random noise has a high weight, then every beam's probability will increase in an equal amount, and every particle's probability will increase, which means more particles will survive during the resampling step. This might, on one hand reduce the precision of the estimation since more particles with less accurate estimated state survived, but on the other increase the possibility for particles to get back on track if something goes wrong, since more particles with different states survived, it is easier to let the correct one survive and kill the wrong one in the following steps.

In order to investigate the weight update regarding longitudinal positions of the particles, we assume that all particles have the lateral position and the heading angle as the ground truth, and only spread out longitudinally. The particle weight update in this case is illustrated in Figure 4.12.

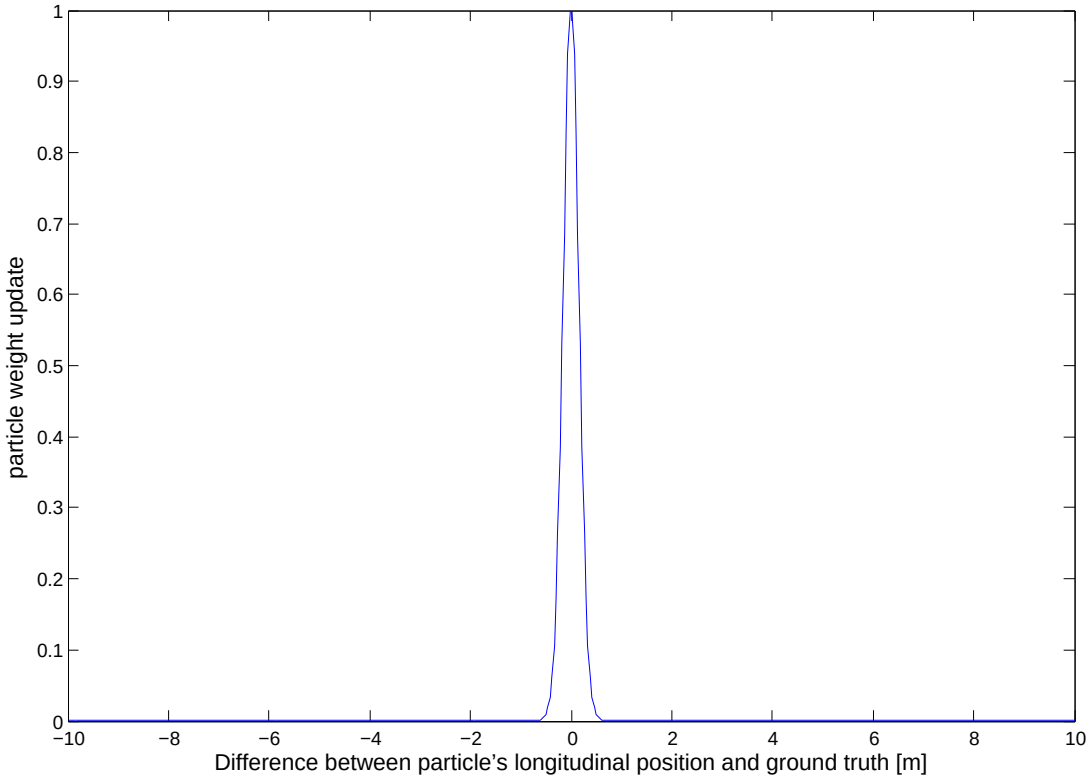

Figure 4.12: Particle weight update according to longitudinal position of the particle, the longitudinal distance between particles and ground truth are evenly spread out from -10 meters to 10 meters, and here the particles that are close to the ground truth gets high weight, while the particles that are far away from the ground truth get very little weights.

It can be seen that if the particle is spot on the ground truth position, then the particle will get the highest weight, and if the particle is far away from the ground truth position it will only get a fairly small weight. The weight update will then influence the resampling process as explained before. The resampling result is shown in Figure 4.13.

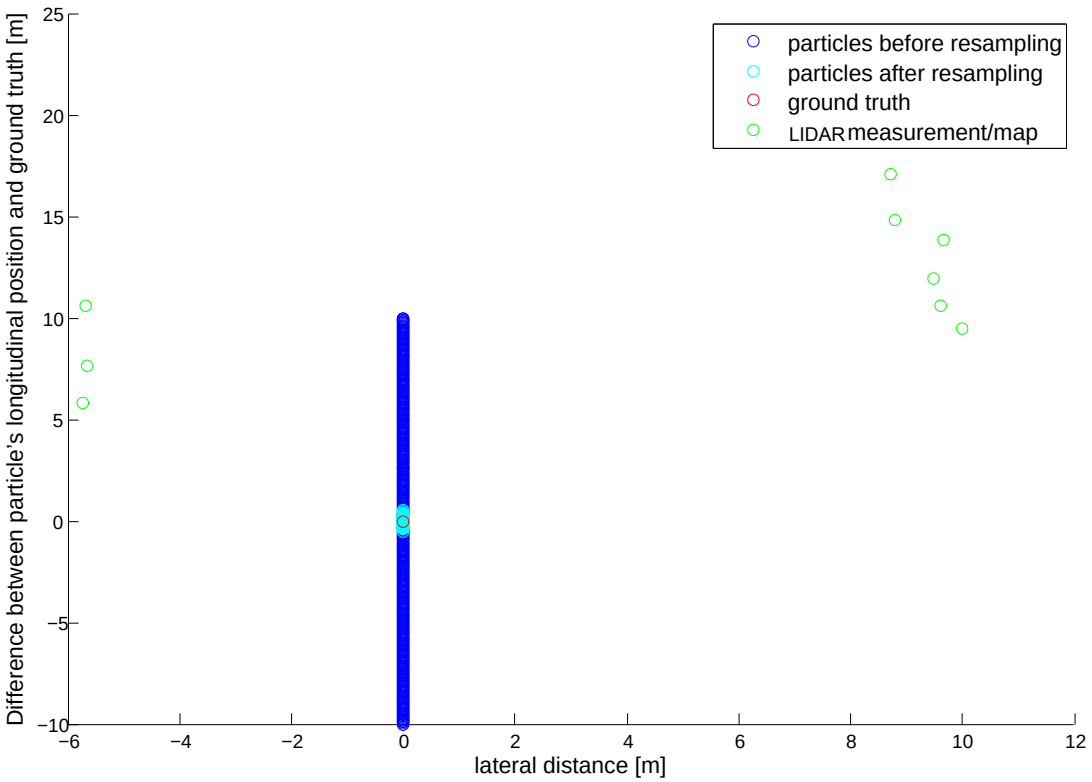

Figure 4.13: Resampling process according to longitudinal position of the particle, the particles are evenly spread out longitudinally from -10 meters to 10 meters, while the ground truth is at 0 position, the graph shows that after resampling, only the particles that are very close to the ground truth got selected.

Figure 4.13 shows that after resampling, the particles with higher weights, i.e. according to Figure 4.12 the particles that are spot on or close to the ground truth have been repeated many times, while the particle with very low weights barely get picked during the resampling process, which is as expected. Similarly, from Figure 4.14 we can see the weight update regarding the lateral positions of the particles. The particles that are located on the expected position get the highest weights, and the particles that are far off the position get almost zero weights. And from Figure 4.15 we can see that the resampling results for particles only spread out laterally. The particles with higher weights has higher chances to get resampled during the resampling process, therefore we can observe that the particles with higher weights shown in Figure 4.14 get resampled many times as Figure 4.15 shows, and the particles with lower weights or almost zero weights barely get resampled.

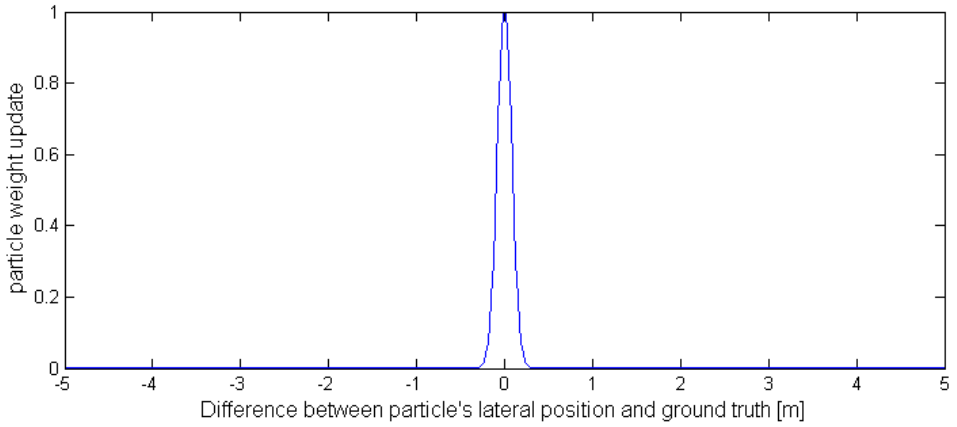

Figure 4.14: Particle weight update according to lateral position of the particle

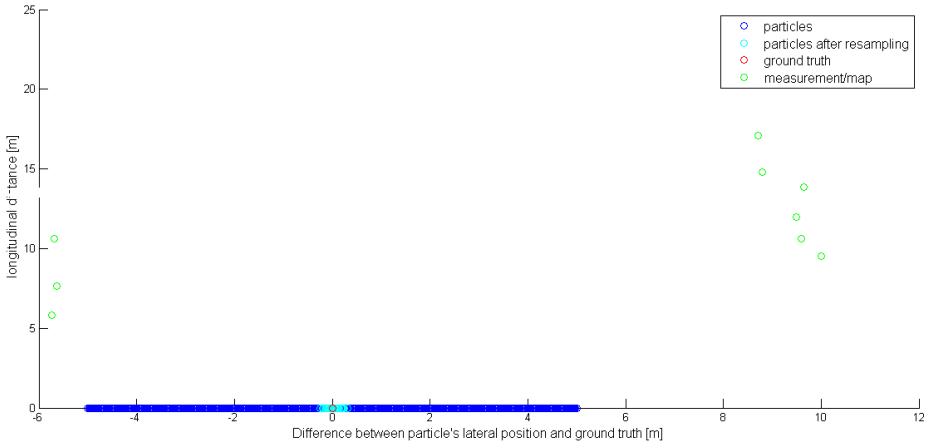

Figure 4.15: Resampling process according to lateral position of the particle, the particles are evenly spread out laterally from -5 meters to 5 meters, while the ground truth is at 0 position, the graph shows that after resampling, only the particles that are very close to the ground truth got selected.

In Figure 4.16 and Figure 4.17 one can see the weight update regarding the heading angles of the particles, and the resampling results for particles with different heading directions.

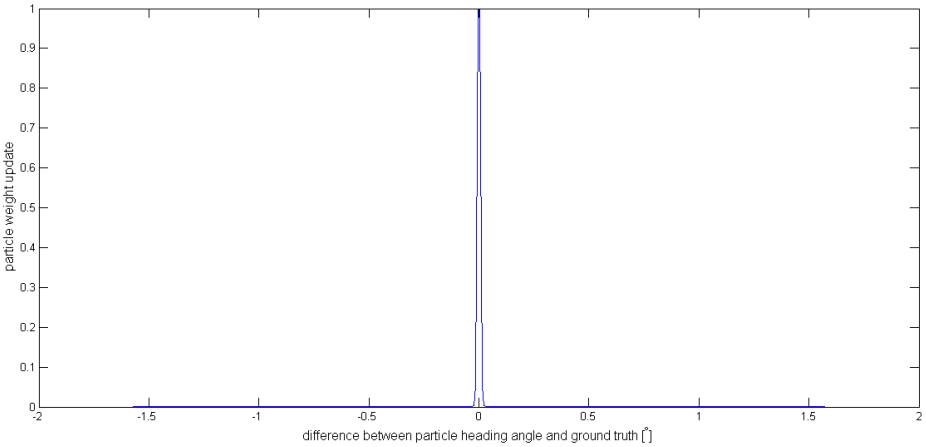

Figure 4.16: Particle weight update according to heading angles of the particle

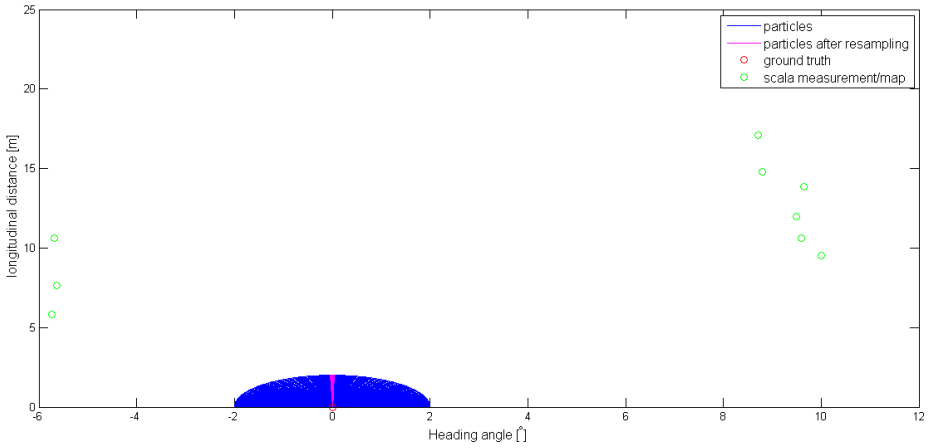

Figure 4.17: Resampling process according to heading angles of the particle

Different combination of z_{hit} and z_{rand}, together with the value of standard deviation of the local measurement noise and the value of random noise will results in different weight update of particles. Optimization regarding these parameters is explained in more detail in Section 4.3.2.

4.2.2.2 Likelihood Field

To calculate the intersection between every beam of every particle and the map takes a lot of computational effort, which therefore slows down the particle filter operation. In order to make the algorithm run faster, another sensor model called likelihood field is tried out. The assumption of the likelihood field model is that the

angle of each beam of the LIDAR is fixed and the distance of each beam for every particle is also fixed. They are all equal to the LIDAR measurement. In this way, the end points of all the beams will have the same position for all particles in each particle's ego view. An illustration of likelihood field is shown in Figure 4.18.

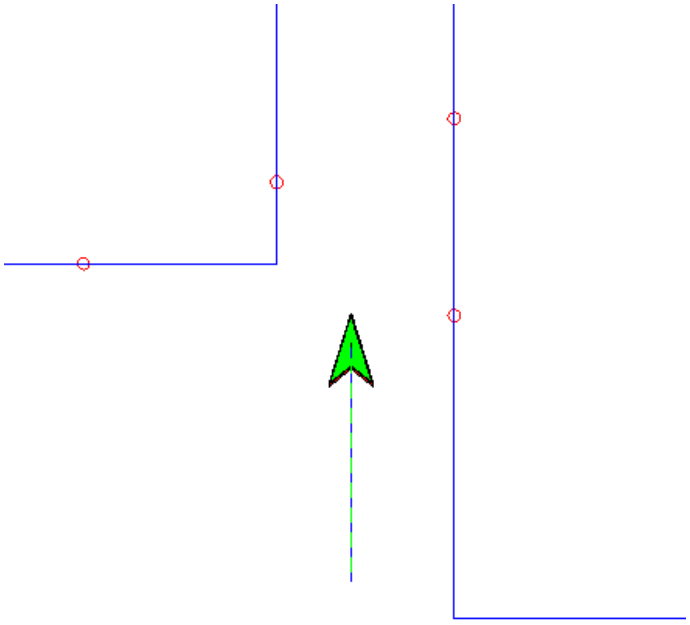

Figure 4.18: Likelihood field model illustration

Here the angle and distance of each beam for every particle is fixed, and they are the same as the LIDAR measurement. In Figure 4.18 one can see the end points of four LIDAR beams from the particle standing in the green object's position. For different particles, as long as the particle's states and the relevant position of the end points of all beams to this particle is known, the position of the end points in the global coordinate frame can be calculated by using a transformation matrix, as Equation (3.3) indicates.

For each beam, the end point can be denoted as $(x_{z_T^k}, y_{z_T^k})^T$, $k = 1, \ldots, K$, where K is the total number of beams. The minimum distance between the end point and an objects in the map m has to be calculated, denoted as $dist$. The probability of each beam is a Gaussian distribution with zero mean regarding local measurement noise, as $p_{hit}(z_t^k|x_t, m) = \mathcal{N}(dist; 0, \sigma_{hit}^2)$. To use the likelihood field model, it is not needed to calculate the intersection between each beam and the map, but only calculating the minimum distance between each end point of the beam and the nearest object in the map is necessary.
Just like the beam-based model, both local measurement noise and random noise needs to be considered. The probability of each beam is $p(z_t^k|x_t, m) = z_{hit} \cdot p_{hit} + z_{rand} \cdot p_{rand}$ and the final probability for each particle can be calculated as Table 3.5 indicates.
Similarly, the optimized combination of z_{hit}, z_{rand}, standard deviation σ_{hit}, and random noise p_{rand} was found by the optimization described in Section 4.3.2.

4.3 Implementation

To make a compact version from the high density point cloud map, a complete solution was implemented and optimized. To localize the car within the desired error margin in the map, a particle filter was implemented with two different sensor models. All functions and algorithms were implemented in MATLAB.

4.3.1 Localization Algorithm

For the particle filter algorithm, both the beam-based model and the likelihood field model have been implemented in MATLAB. The process of each iteration of the particle filter is explained in Figure 4.10.

In the prediction step, the predicted state is calculated based on the prior state and control input, the algorithm shown in Equations (4.1) to (4.3) is implemented. For the update step, the predicted state is updated by taking the map and LIDAR measurement into consideration. Both the beam-based model and the likelihood model are implemented as particle weight update methods. For the beam-based model, the algorithm shown in Table 3.4 is implemented. And for the likelihood field model, the algorithm described in Table 3.5 is implemented.

For parameters such as the weight of local measurement noise z_{hit} and random noise z_{rand}, standard deviation of the Gaussian local measurement noise distribution of σ_{hit}, random noise p_{rand}, number of particles, and number of selected beams, initial values has been set in the beginning to test the algorithm. After the implementation had been confirmed functioning well, many different combinations of those parameters has been tried out in order to find out to which degree each parameter settings can influence the result of the particle filter. An optimization algorithm has also been implemented, which is explained below.

4.3.2 Optimization of Localization Algorithm Parameters

In order to make the localization algorithm perform as good as possible, the tunable parameters of the algorithm were optimized. The possible parameters to optimize were the amount of Gaussian noise in the particle motion model determined by r_{speed}, r_{yaw} and C_{yaw}, the variance of the likelihood field in the sensor model $sigma_{hit}$, the probability of random noise in the sensor model P_{rand} and the weights to weigh together the probabilities in the sensor model. The weights for the probabilities were considered as only one parameter since both weights must sum up to 1, giving $z_{hit} = 1 - z_{rand}$, giving a total of 6 parameters to optimize.
The optimization was performed by using the Twiddle algorithm described in Section 3.9. The cost function to be minimized by the optimization algorithm was constructed to put equal weight on all three states to estimate; lateral- and longitudinal position and heading angle. Since the error in longitudinal position was so much larger than for lateral position and heading angle the error for the two latter had to be emphasized. The reason for this was that improvement in longitudinal

positioning was not be made on the cost of making worse results in lateral position or in heading angle. The cost function was calculated as

$$\begin{aligned} \text{cost} = & 20 \cdot \mu_{\text{MaxLateralError}} + 40 \cdot \mu_{\text{MeanLateralError}} + \\ & \mu_{\text{MaxLongitudinalError}} + \mu_{\text{MeanLongitudinalError}} + \\ & 3 \cdot \mu_{\text{MaxHeadingError}} + 9 \cdot \mu_{\text{MeanHeadingError}} \end{aligned} \tag{4.4}$$

where μ denotes the average over all analyzed log-files. Due to time constraints only 6 minutes of driving from log-files were used in the optimization in order for it to finish in reasonable time.

5
Results

5.1 Map Extraction

Implementation of the method for automatic map extraction described in Section 4.1 was made and an example of the result can be seen in Figure 5.1. In this figure it can be seen that the extracted map lines in red over the high density point cloud barriers in green. This figure is only a small part of the map but it is representative for the overall result.

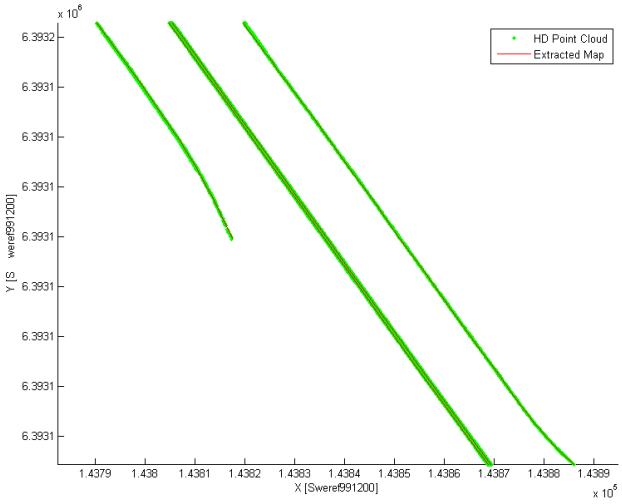

Figure 5.1: An example of the result of the automatically extracted map. The green points are the barriers from the high density point cloud and the red lines are the resulting map. Note how well the extracted map lines represents the point cloud.

5.1.1 Barrier Classification

The results from the forward selection of feature parameters for barrier classification can be seen in Figure 5.2. The x-axis of the plot is cumulative, meaning that the misclassification result for each feature is for that feature combined with all the previous ones. The chosen features for further use in the classification process where the ones that gave the lowest misclassification rate; y-variance, z-variance and xz-covariance. Cross-validation on the training set gave an average misclassification

rate of 3% using these parameters. As can be seen in the result plot below, adding more parameters for classification increased the misclassification rate.

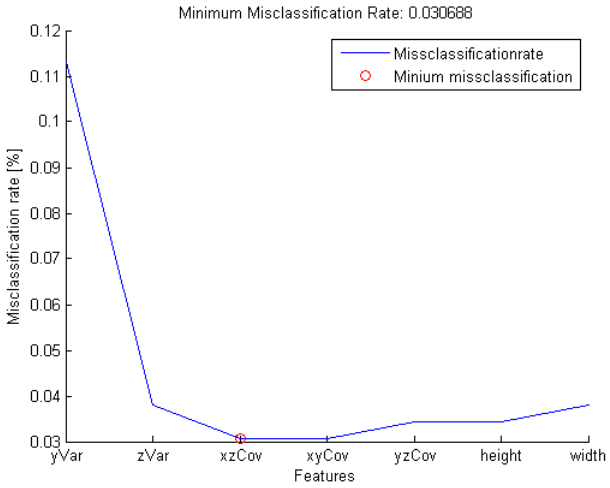

Figure 5.2: The results from the forward selection of feature parameters. The line shows the average misclassification rate for every added feature.

5.2 Compact Map Representation

The result of the compact map representation can be measured in two parameters; amount of data per kilometre road and how well the localization algorithm performs when using the map. The amount of data per kilometre road is a number easy to calculate, whereas the performance of the map for localization is harder to measure since the localization depends on many parameters. For this reason the performance of the map for localization will be the result of the positioning algorithm.

In a worst case scenario where there are two barriers alongside the road, each of them being curved, requiring one point per meter road to be described accurately, the compact map will need 32 kB/km storage. This is calculated as: 1000 line segments per barrier, having 1001 points with double precision x and y for every point, times two barriers resulting in

$$\begin{aligned} \text{mapDataSize/km} &= 1001 \text{ points } \cdot 2 \text{ coordinates } \cdot 8 \text{ bytes } \cdot 2 \text{ barriers} \\ &= 32032 \text{ bytes/km.} \end{aligned} \tag{5.1}$$

However, road side barriers are straight most of the time allowing the map representation to be compressed even further by removing redundant point according to the algorithm described in Table 4.1. This algorithm removed on average 70% of the points in the barriers reducing the data size of the map representation to 9.6 kB/km.

5.3 Localization of the Vehicle

The localization simulation was run using the compact map with recorded log-data from 23 log-files, corresponding to approximately 23 minutes of driving. Results of average errors and maximum errors can be seen in Table 5.1 and Table 5.2. The algorithm was run with the optimized parameters in Table 5.3, 500 particles for the particle filter and with 10 beams/points chosen from every LIDAR scan for localization. The evaluation was done using both raw scan data as well as the filtered scan data, for comparison between the two. Results from individual log-files can be seen in the sections below.

Average positioning errors

	Raw scans		**Filtered scan data**	
	Error	**SD**	**Error**	**SD**
Average lateral error	0.043 m	0.041 m	0.035 m	0.026 m
Average longitudinal Error	0.66 m	0.57 m	0.48 m	0.37 m
Average heading angle error	0.14°	0.13°	0.12°	0.091°

Table 5.1: Average lateral, longitudinal, and heading angle error with standard deviations. Raw LIDAR data compared with filtered scan data

Maximum positioning errors

	Raw scans	**Filtered scan data**
Maximum lateral error	0.88 m	0.13 m
Maximum longitudinal error	2.4 m	1.9 m
Maximum heading angle error	2.4°	0.79°

Table 5.2: Maximum lateral, longitudinal, and heading angle error using raw scans and filtered scan data

5.3.1 Lateral Error

In Figure 5.3 and Figure 5.4, one can see results for the individual log-files with a comparison between using the raw LIDAR scans and using the filtered scan data in the localization algorithm.

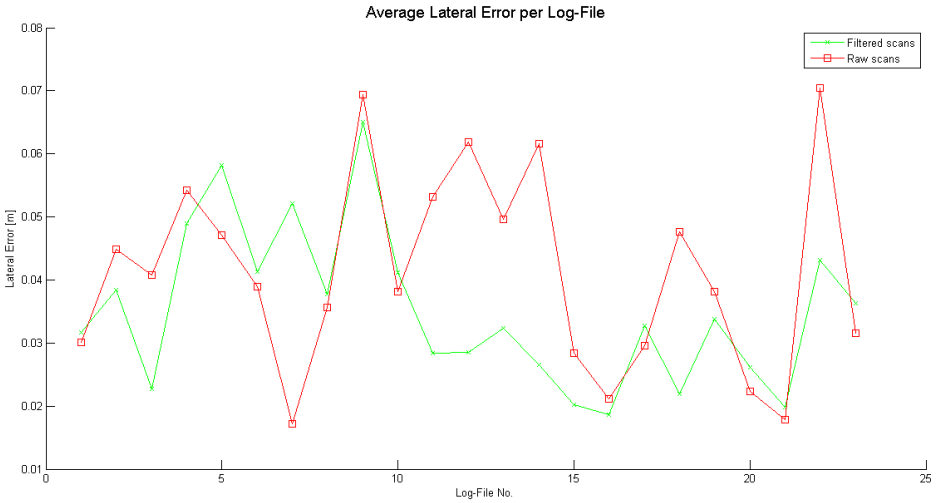

Figure 5.3: A comparison over the average lateral error in each log-file, when using raw scan data and using the filtered scan data. Note that the lateral error from using the filtered scan data is most of the time lower or approximately equal to lateral error when using raw scans.

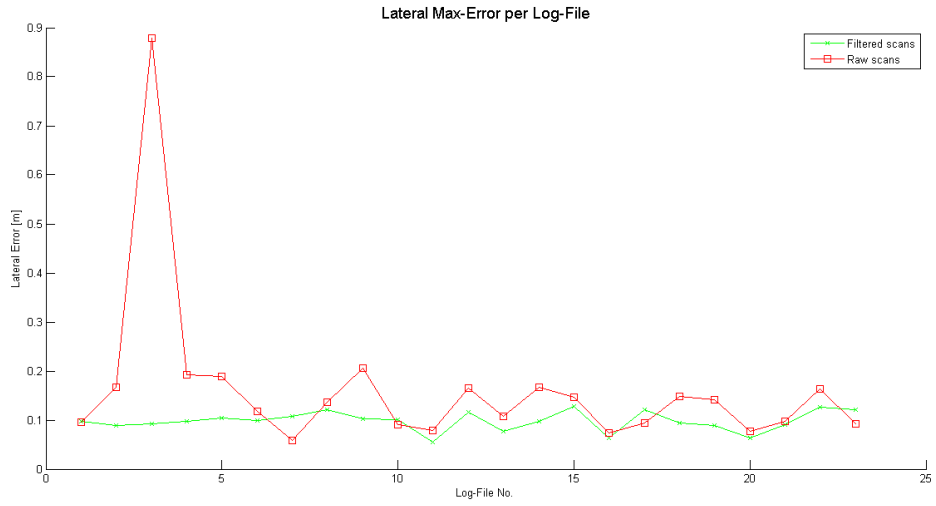

Figure 5.4: A comparison over the lateral max-error in each log-file, when using raw scan data and the filtered scan data. Note that the lateral max error from using the filtered scan data is most of the time lower or approximately equal to lateral max error when using raw scans.

5.3.2 Longitudinal Error

In Figure 5.5 and Figure 5.6, one can see results for the individual log-files with a comparison between using the raw LIDAR scans and using the filtered scan data in the localization algorithm.

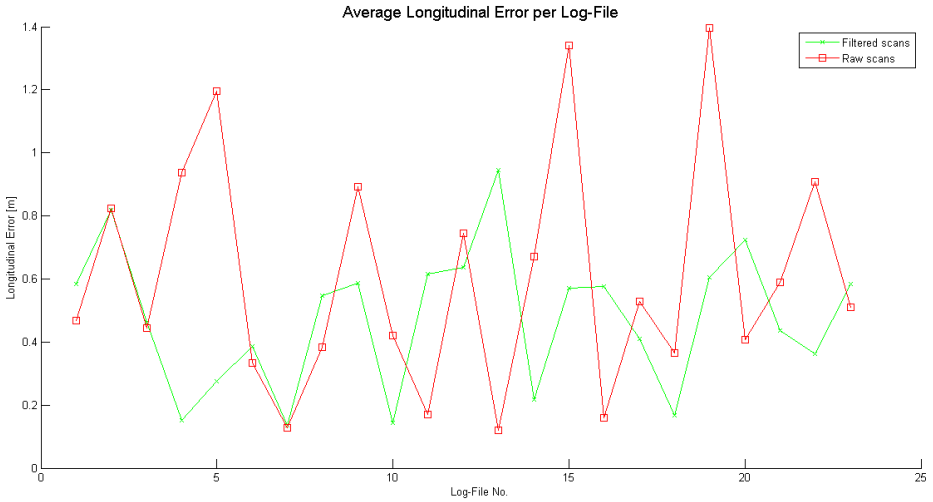

Figure 5.5: A comparison over the average longitudinal error in each log-file, when using raw scan data and the filtered scan data. Note that, on average, the filtered scan data gives a lower average error than the raw scans.

Figure 5.6: A comparison over the longitudinal max-error in each log-file, when using raw scan data and the filtered scan data. Here there are no obvious difference between raw scans and the filtered scan data, however, they do perform differently on the same log-files.

5.3.3 Heading Error

In Figure 5.7 and Figure 5.8, one can see results for the individual log-files with a comparison between using the raw LIDAR scans and using the filtered scan data in the localization algorithm.

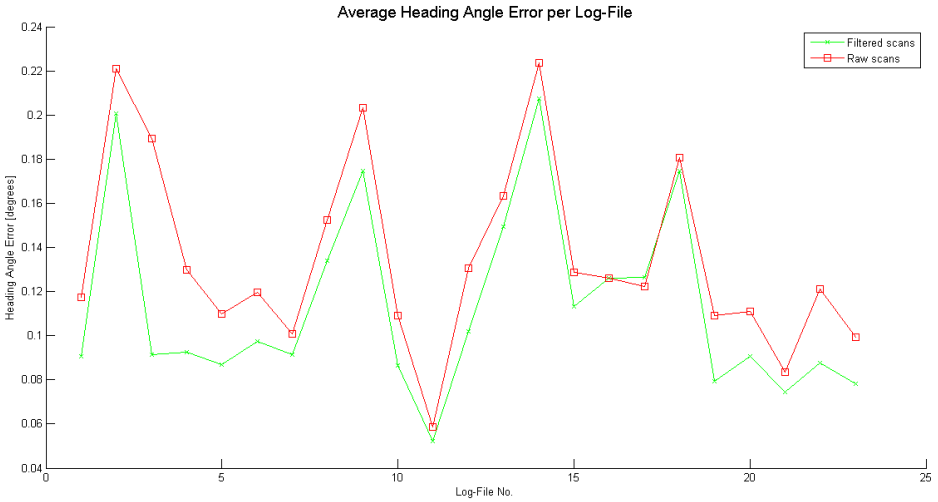

Figure 5.7: A comparison over the average heading angle error in each log-file, when using raw scan data and the filtered scan data. Note how the filtered scan data performs better for almost every log-file, however, differences are only about 0.05°.

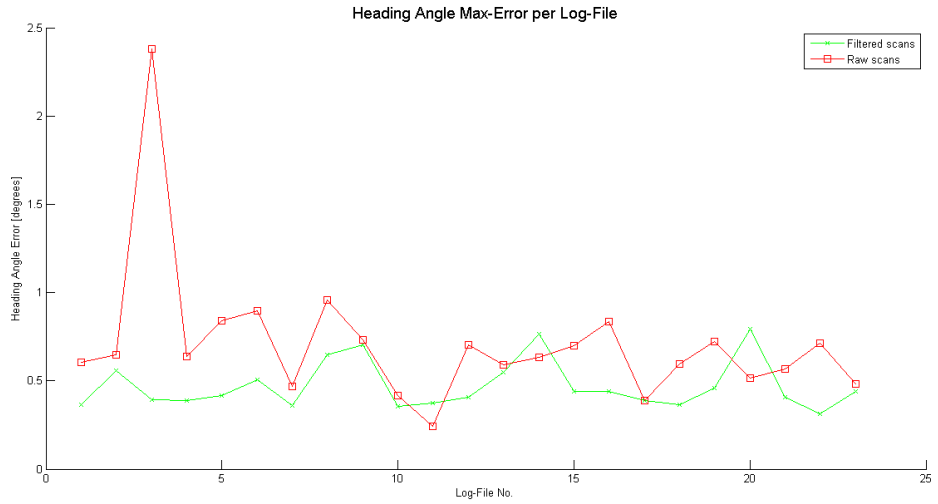

Figure 5.8: A comparison over the heading angle max-error in each log-file, when using raw scan data and the filtered scan data. Note that the filtered scan data most of the time performs better or equal to when using raw scan data.

5.4 Optimization of Localization Algorithm Parameters

By running the twiddle algorithm on the tuneable parameters of the localization algorithm, the results in Table 5.3 were achieved. The upper part is the initial guess of parameters that were chosen by reasoning, the lower part is the resulting parameters after optimization. The cost function described in Equation (4.4) was evaluated to 17.8 for the initial guess parameters and 10.4 for the optimized parameters, indicating the magnitude of the improvement of the localization algorithm. In the sections below one can see the results of the optimization for each individual log-file.

	r_{speed}	r_{yaw}	$C_{yawNoise}$	$\sigma_{likelihood}$	p_{rand}	z_{hit}	z_{rand}
Initial guess	0.1	0.1	10	0.05	0.1	0.9	0.1
Optimized	0.02	0.25	5	0.1	0.6	0.7	0.3

Table 5.3: The tuneable parameters of the localization algorithm; the initial guess of the parameters and the resulting parameters after twiddle optimization,

5.4.1 Lateral Error

In Figure 5.9 and Figure 5.10, one can see results for the individual log-files before and after the optimization of parameters. Since the optimization was carried out using the filtered scan data, the results below are produced using the filtered scan data as well.

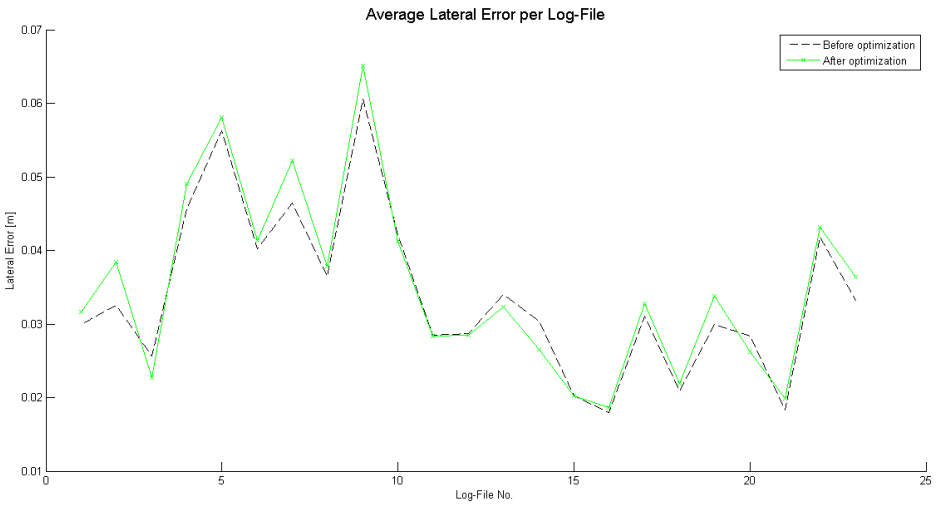

Figure 5.9: A comparison over the average lateral error in each log-file, before and after optimization of the algorithm parameters. Note how the average lateral error became slightly worse after optimization.

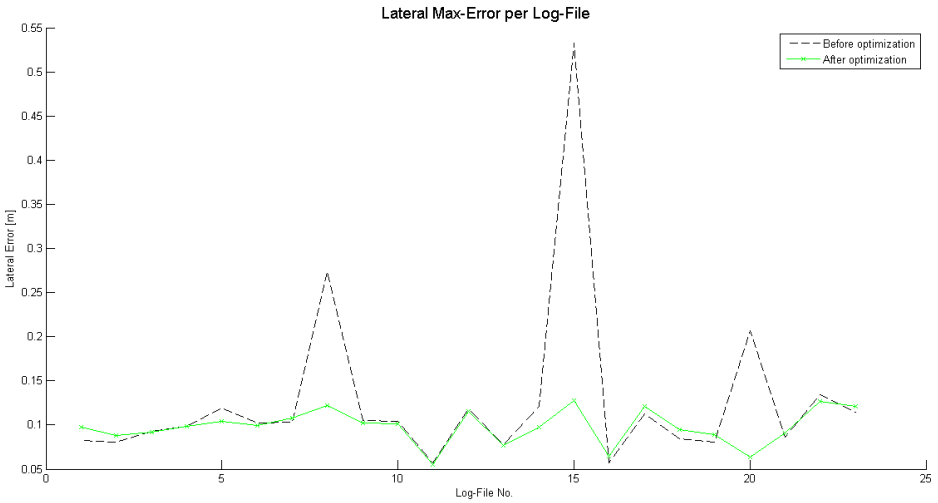

Figure 5.10: A comparison over the lateral max-error in each log-file, before and after optimization of the algorithm parameters. Note how the largest peaks in the lateral max error has been drastically reduced after optimization.

5.4.2 Longitudinal Error

In Figure 5.11 and Figure 5.12, one can see results for the individual log-files before and after the optimization of parameters. Since the optimization was carried out using the filtered scan data, the results below are produced using the filtered scan data as well.

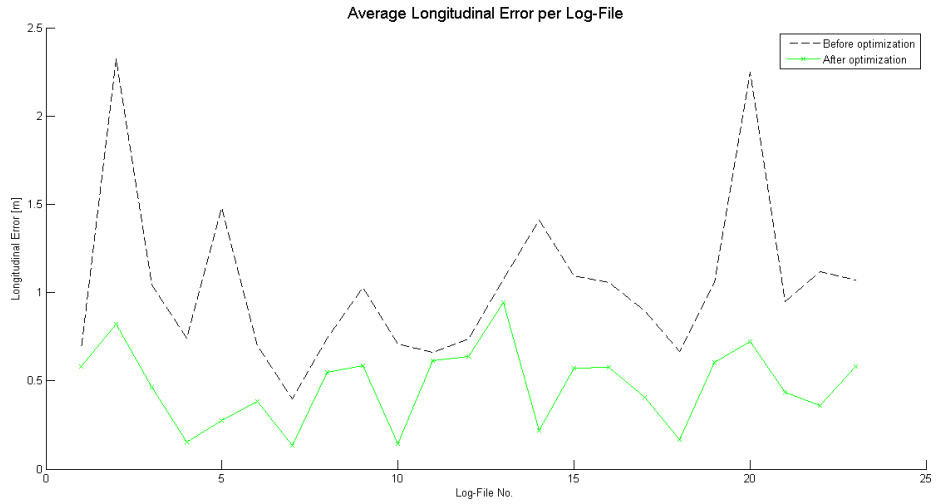

Figure 5.11: A comparison over the average longitudinal error in each log-file, before and after optimization of the algorithm parameters. Note how the average longitudinal error has been reduced for every log-file, after optimization.

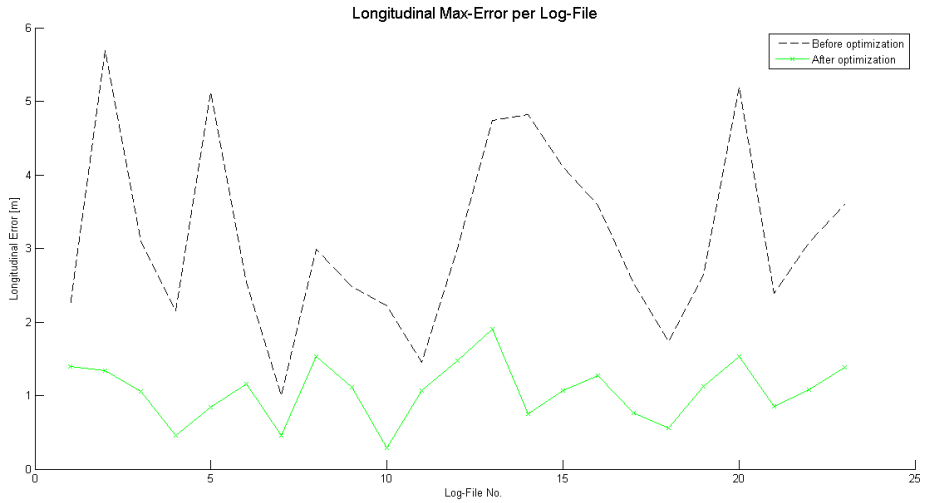

Figure 5.12: A comparison over the lateral max-error in each log-file, before and after optimization of the algorithm parameters. Note how the maximum longitudinal error for each log-file is reduced after optimization.

5.4.3 Heading Error

In Figure 5.13 and Figure 5.14, one can see results for the individual log-files before and after the optimization of parameters. Since the optimization was carried out using the filtered scan data, the results below are produced using the filtered scan data as well.

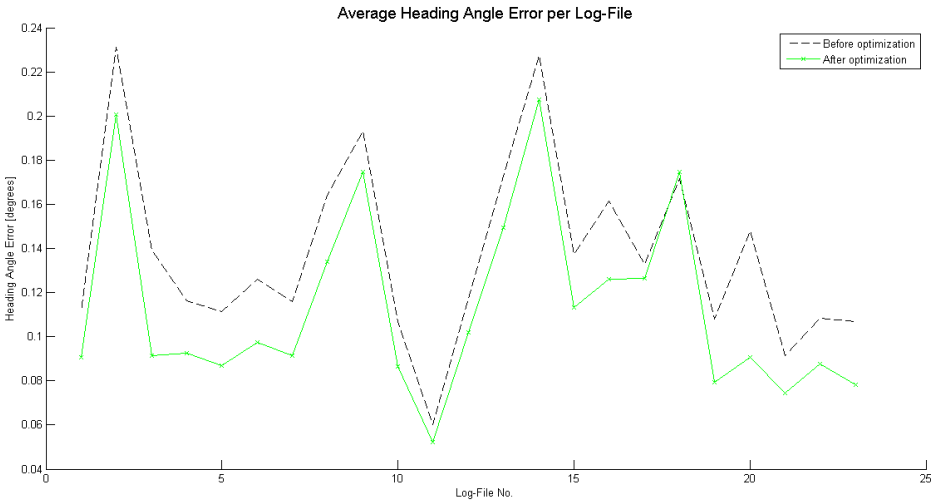

Figure 5.13: A comparison over the average heading angle error in each log-file, before and after optimization of the algorithm parameters. Note how the average heading error is slightly better or equal for every log-file, after optimization.

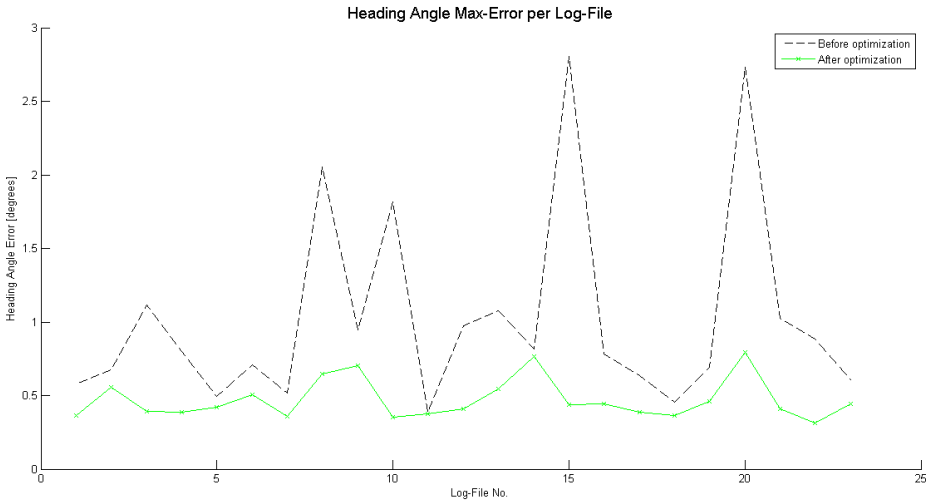

Figure 5.14: A comparison over the heading angle max-error in each log-file, before and after optimization of the algorithm parameters. Note that the optimization improved the heading max-error significantly for most log-files.

6
Discussion

The overall impression from the results is that the project have been successful in both automatically extracting a map from the high density point cloud and storing the map in a compact representation, as well as being able to accurately position the car in the map by using the front facing LIDAR. In Table 5.1 and Table 5.2 one can see that lateral position and heading angle are estimated very precisely when using the filtered scan data, with both estimates having a very low mean error as well as a maximum error that is low enough to ensure that the vehicle stays inside the road lane. The longitudinal position has a fairly low mean error and a max error that is not terrible. Although there is still room for improvements, the map could be used in a cloud-based fashion, and the positioning results are enough for autonomous driving.

6.1 Map

The map extraction seems to be working well with no major errors or missed barriers. Depending on the point cloud and classification algorithm there are always a risk for misclassification of segments or barriers standing so close to each other that they end up in the same cluster leading to errors in position and/or misclassification. The algorithm handles these errors by making the corrections explained in Section 4.1.5, however, there is no guarantee that it will be sufficient and manual inspection/verification will still be needed. An idea for a different approach for determining a robust position of barrier segments than taking the average position, would be to find a point that lies on the edge of the barrier about the height were the LIDAR would "see" the barrier. This is a more complicated point to calculate but it might make classification and offset correction of the barriers redundant, leaving less possible sources of error.

6.2 Localization

From Table 5.1 and Table 5.2, one can see that the filtered scan data gives better position estimates. This is expected though, since the filtered scan data contains dynamic and static object classification, allowing the possibility of filtering out most points that belong to dynamic objects.

The numbers in Table 5.1 and Table 5.2 indicate that the localization algorithm is working very well. One should keep in mind though, that the results are only from

about 23 minute of driving. To be able to say that it works in all traffic situations one has to verify it by running simulations on a lot more log-files.

6.2.1 Lateral Error

As one can see in Table 5.1 and Table 5.2, the average lateral error using filtered scan data is 3.5 cm, which is within our goal of 5 cm. The maximum lateral error is 0.13 m, which is about half a tyre width, leaving plenty of margin on both sides of the car when driving in the middle of the lane. This is a reasonable result since the barriers contain a lot of lateral information easy to perceive for the LIDAR. According to the LIDAR specification the distance resolution is $< 0.1m$ [17], and our results indicates that it can perform range measurements with centimetre accuracy. The lateral performance of the localization algorithm could be improved in many ways: optimize the offsets for each barrier type by an automatic algorithm like e.g. twiddle; use vertical information of the barrier; including more objects to the map such as bridge pillars and light poles.

6.2.2 Longitudinal Error

The average longitudinal error seen in Table 5.1 is 0.48 m, which is within our goal of 0.5 m. Since the route of the log file contains much more straight road than curvy road, the longitudinal information might be a bit insufficient for the positioning, as the barriers position on both sides of the road differs little in a straight road. Nevertheless, both the average and maximum longitudinal error (1.9 m, shown in Table 5.2) are within the acceptable range. Another aspect to take into account is that longitudinal position is much harder to estimate than the lateral, due to the fact that the longitudinal speed of the vehicle is many times larger than the lateral speed, giving a higher uncertainty in the longitudinal direction. Besides, the lateral information is much richer than longitudinal information in this case.

6.2.3 Heading Angle Error

Table 5.1 and Table 5.2 shows that the average heading angle error using filtered scan data is 0.12° with a maximum error of 0.79°. These results would seem adequate for autonomous driving and they are probably the product of having several LIDAR measurements on the same barrier but at different distances, giving pose hypothesises with incorrect heading angle a lot lower weight, since measurements on the barrier far away will be very off, even for small angles.

6.3 Particle Filter

The particle filter contains two stochastic processes; first when the particles are propagated forward using the measured yaw-rate and speed with added random Gaussian noise for every particle and second, in the re-sampling step where a new set particles are drawn randomly from the existing set, picking a particle for the new set with a probability proportional to its weight. This results in that two

simulations on the same log-file will not produce the exact same results. Due to the computational limit of the computer, the following settings were chosen for the simulation: 500 particles, 10 beams, and 0.1 likelihood field standard deviation $\sigma_{likelihood}$. The result of each simulation depends on how wide the particles spread if the map goes off slightly or when other cars might be blocking the view of the LIDAR. When $\sigma_{likelihood}$ is large the particles spread out more, therefore the particles can correct their positions quicker if the estimated position is wrong, on the other hand they have higher risk of creating an incorrect position estimate when they spread out wider. One can see that there is a trade- off between precise accuracy and the possibility of correcting the position estimate if going off too far. The optimized value of $\sigma_{likelihood}$ depends on which result is more important, for example to get the least average lateral error and the least maximum lateral error the setting of $\sigma_{likelihood}$ is different. The optimized value in Table 5.3 is only calculated under the assumption that roughly equal emphasis has been put on both average and maximum lateral error, and less focus on longitudinal and heading angle error, as the cost function Equation (4.4) indicates.

6.4 Parameter Optimization

In Figure 5.9 to Figure 5.14 one can see that the optimization managed to improve all results except the average lateral error which increased slightly in almost every log-file. This increase, though, is on millimetre level and is justified by quite big improvements in all other errors. For example Figure 5.10 shows that the peak of the maximum lateral error has been reduced for about 0.5 meter after optimization.

Table 5.3 indicates how the optimization results show certain modifications of parameter settings compared with the initial guess. For the noise that has been added on the speed $r_{speed} \cdot speed$, the optimized value of r_{speed} is 0.02, which is much smaller than the original guess 0.1. This indicates that the speed sensor is quite accurate and a smaller longitudinal spread among the particles is enough to cover possible longitudinal states

As for the noise that has been added on the yaw, it can be seen from Table 5.3 that the optimized value of r_{yaw} is bigger and the constant noise of yaw rate $C_{yawNoise}$ is smaller than the initial guess. This makes sense since higher r_{yaw} value will make the particles spread more when the yaw rate is higher and vice versa, which makes the particles propagate more according to the maneuver of the car, i.e. if the car is turning more the particles will spread more in terms of heading angles and if the car is going straight the particles will spread less in heading angles.

The standard deviation of the likelihood field σ_{hit} is doubled after the optimization. This will make more particles have higher weights in the update step. The probability of random noise p_{rand} was also increased a lot after optimization. This means that the weights of all the particles are likely to be increased. The weight on random noise z_{rand} increased from 10% to 30%, which also makes the particles more indifferent from one another. While the particles whose LIDAR measurements fit better in

the map will get higher weight, other particles will also get some weight depending on the value of random noise. This result is a bit surprising, since it is assumed that the particle filter might perform better when the particles that are off get eliminated in time. One possible explanation could be that the slightly off particles will not get killed so easily during the resampling stage but particles whose measurements fit better in the map are still more likely to get selected during resampling process. The optimization here made more particles survive the resampling process, which in turn will give a greater diversity in the state hypothesises among the particles.

6.5 Conclusion

The conclusion is that the method for automatic map extraction is working satisfactory, but it still needs manual verification in its current state. The storage size of the map representation is low enough to allow cloud based storage. The localization algorithm based on a particle filter and a likelihood field sensor model works very well and is robust to surrounding traffic. According to the 23 log files that have been tested, using the filtered scan data instead of raw measurements improved the localization results to some degree, due to the filtered scan data filteres out the dynamic objects and ground information. If the results can be verified, they are sufficient for autonomous drive.

7

Future Work

7.1 Look-up table of the likelihood field

The method to calculate the likelihood field in this thesis is that first calculate the position of the end point of a beam, then calculate the minimum distance between this point and the lines in the map, after that find out the possibility of this distance in a zero-mean Gaussian distribution. This process is time consuming due to that every beam of every particles need to be calculated in these three steps.

One possible method to speed up the calculation could be create a look-up table for the likelihood field. With a look-up table, one just need to calculate the end point of each beam, then check the possibility in the look-up table based on the position of this point. This could potentially save some operation time for every time stamp. To make the loop-up table, the map of the road need to be converted into a pixel presentation, and each pixel represents a possibility according to its distance to the barriers in the map. In this thesis a set of pixel map has been made, an illustration is shown in Figure 7.1, and a zoom-in image of a part of a barrier in shown in Figure 7.2.

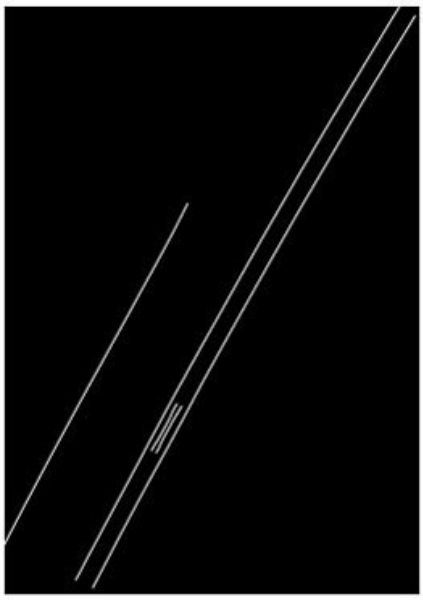

Figure 7.1: Map in pixel representation, serves as a likelihood loop-up table, possibility is represented by the brightness level of the pixel, the brighter the possibility is higher. The left line in the picture is a barrier on the side of the road, the two long lines on the right are the central barrier, and two short lines on the right is a bridge pillar.

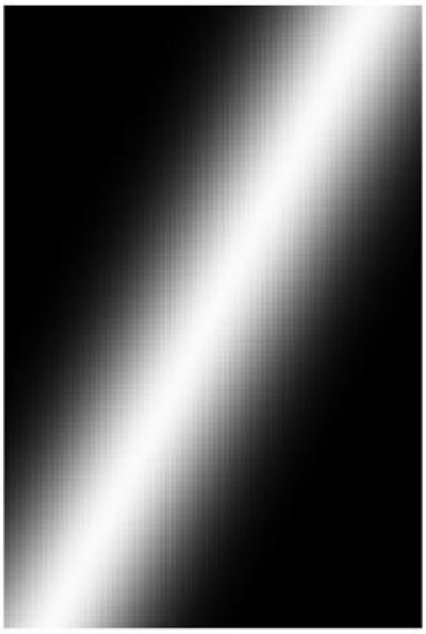

Figure 7.2: Map in pixel representation, a zoom in version of a part of a barrier. Possibility is represented by the brightness level of the pixel, the brighter the possibility is higher.

The several log files has been tested using the likelihood loop-up table, while for some log files it works for others some issues still need to be investigated further. For example if each pixel map covers the whole longitudinal and lateral range of a 100-meter long barriers segment, then the overlap area between maps need to be determined with careful consideration in order to make sure all end points of beams of all particles will be covered in the map. A more reliable solution of the pixel representation of the map need could be developed in the future.

7.2 False measurement tolerance

When the view of the LIDAR mounted in front of the ego vehicle has been completely blocked by a truck or bus, the measurement update might lead to false estimation. Therefore, the next step would be to make the algorithm detect the false barrier measurement, skip the measurement update in these time stamps and only produce estimation position by dead reckoning. For example, if the LIDAR detects that the distance to barrier remains the same at one side, but suddenly becomes much shorter at the other side, it might be blocked by a bus or truck. In that case only the prediction step of the particle filter will be performed, the measurement will not be taken into consideration, since it is a false measurement. This might prevent errors in the estimation of position.

7.3 Implementation of the localization algorithm

The localization algorithm in this thesis is implemented in MATLAB only. The possible ways to speed up the algorithm in order to be used in the real time situation could be implementing the algorithm in C or C++, and using parallel computing. The particle filter is a highly parallelizable algorithm. The heaviest computational part is calculating the weight for each particle. This calculation is exactly the same for each particle and could easily be distributed over several cores, with a possibility of improving the efficiency of the algorithm greatly. Switching programming language from MATLAB to C could most likely also improve the performance since C allows for the programmer to control so that the memory is used in a more efficient way than MATLABs automatic memory handling.

Bibliography

[1] T. COWEN, "Can i see your license, registration and c.p.u.?" http://www.nytimes.com/2011/05/29/business/economy/29view.html, 2011.

[2] S. Thrun, D. Fox, and W. Burgard, *Probabilistic Robotics*. The MIT Press, 2005.

[3] W. Burgard, D. Tipaldi, M. Ruhnke, and B. Steder, "Introduction to mobile robotics." [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ss15/robotics/slides/07-sensor-models.pdf

[4] K. et al., "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 24, no. 7, pp. 881–892, July 2002.

[5] B. S. E. et al., *Cluster Analysis*, 5th ed. John Wiley & Sons, Ltd, ISBN: 9780470749913, 2011.

[6] "Kartprojektioner: Sweref99." [Online]. Available: https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Referenssystem/Tvadimensionella-system/SWEREF-99-projektioner/

[7] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[8] S. Thrun, D. Fox, and W. Burgard, *Probabilistic Robotics*. The MIT Press, 2005, ch. 2, p. 26.

[9] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," pp. 182–193, 1997.

[10] D. Hong-de and D. Shao-wu, "Performance comparison of ekf/ukf/ckf for the tracking of ballistic target," *Telkomnika Indonesian Journal of Electrical Engineering*, vol. 10, no. 7, pp. 1537–1542, November 2012.

[11] G. Einicke and L. White, "Robust extended kalman filtering," *Signal Processing, IEEE Transactions on*, vol. 47, no. 9, pp. 2596–2599, Sep 1999.

[12] S. Thrun, D. Fox, and W. Burgard, *Probabilistic Robotics*. The MIT Press, 2005, ch. 3, p. 43.

[13] ——, *Probabilistic Robotics*. The MIT Press, 2005, ch. 4, p. 96.

[14] ——, *Probabilistic Robotics*. The MIT Press, 2005, ch. 6, pp. 153–172.

[15] D. Barber, *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[16] H. Liu and H. Motoda, *ComputationalMethods of Feature Selection*, 1st ed. CRC Press Taylor & Francis Group, ISBN: 9781584888789, 2007.

[17] *Technical Characteristics*, Ibeo Automotive Systems GmbH. [Online]. Available: http://autonomoustuff.com/ibeo-scala/

A

Appendix

A.1 Barrier Types

Along the Gothenburg demo route there are several types of different man-made roadside barriers. To be able to reference the different types in a consistent manner, the following types and names are suggested. All types are visualized with a picture and a plot of how they appear in the high density point cloud accompanied by a short description.

Elevated Rail
This is one of the most common barrier types along the demo route. It is a metal rail mounted on poles in the ground.

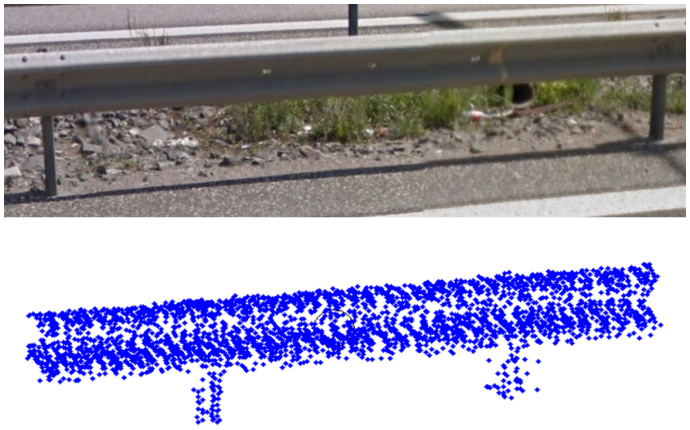

Figure A.1: An example of an elevated rail and its point cloud representation.

Offset Rail
The offset rail has the same rail type as the elevated rail but is is mounted on vertical poles that connect to the poles in the ground, creating a space in between the rail and the ground poles, hence the name.

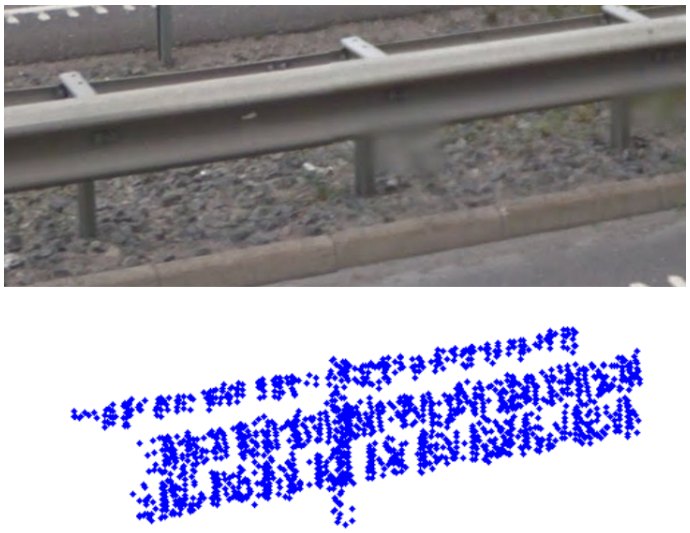

Figure A.2: An example of an offset rail and its point cloud representation.

Double Rail

The double rail consists of two rails mounted on each side of a center pole. This rail type is commonly used as a center barrier dividing the road into two different driving directions.

Figure A.3: An example of a double rail and its point cloud representation.

Protected Rail

This is an elevated rail but with some kind of fencing behind it. This rail type is commonly used over bridges.

Figure A.4: An example of a protected rail and its point cloud representation.

Pipe Rail
This barrier type has two pipe-shaped rails mounted in parallel on poles standing in the ground.

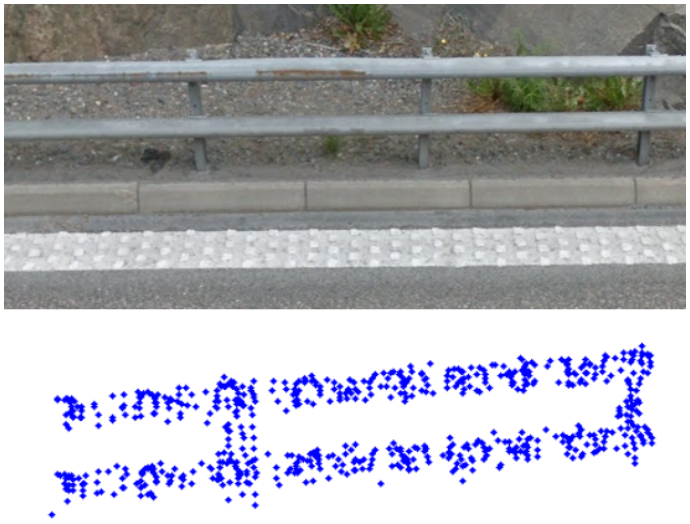

Figure A.5: An example of a pipe rail and its point cloud representation.

Concrete Cone
The concrete cone barrier is normally a center barrier separating traffic going in different directions. It consist of cone shaped concrete blocks that are connected with each other.

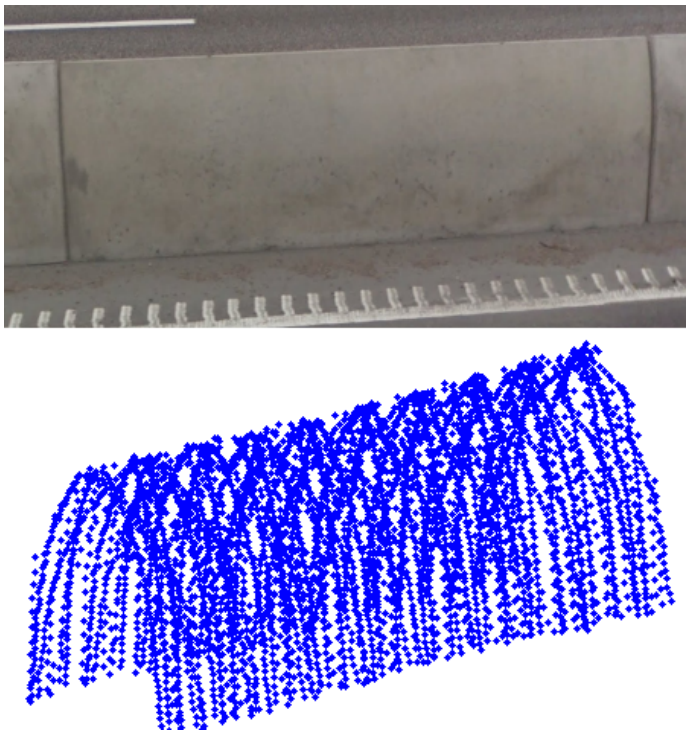

Figure A.6: An example of a concrete cone and its point cloud representation.